# Scalable and Fast Method for IoT Malware Clustering

何, 天祥



出版情報：九州大学, 2022, 博士（工学）, 課程博士
バージョン：
権利関係：

Kyushu University

Doctoral Thesis

# Scalable and Fast Method for IoT Malware Clustering

*Author:*
Tianxiang He

*Supervisor:*
Prof. Jun'ichi Takeuchi

Department of Information Science and Technology
Graduate School of Information Science and Electrical Engineering, Kyushu University

February 3, 2023

# Contents

# Chapter 1

# Introduction

IoT malware has been a rapid proliferation in recent years, and a large amount of malware is newly spread every day. Honeypots and anti-malware platforms (such as VirusTotal [1]) collect large amounts of malware samples to keep track of popular malware trends. By analyzing these malware samples, it is possible to know the threat of malware to cyberspace. To analyze the massive amount of malware specimens, efficient malware analysis methods are required.

Clustering is a compelling method to analyze these large-scale malware sets efficiently. In this paper, we focus on clustering based on a phylogenetic tree of malware specimens and discuss efficient methods of constructing phylogenetic trees. We have a particular interest in malware's phylogenetic tree because it can not only cluster malware but also investigate the evolutionary relationships of malware specimens. However, conventional methods to construct a phylogenetic tree are very time-consuming when facing a large malware set. Therefore, we propose a new phylogenetic tree construction method from a large-scale malware specimen set Our method is outlined as follows: we first calculate the distances between malware specimens and then use these distances to create a phylogenetic tree. For clustering, we divide the phylogenetic tree into appropriate subtrees, each of which is a cluster.

We use the normalized compression distance (NCD) to measure the similarity (distance) between execution-type malware binaries. NCD does not require domain knowledge and can be applied to many file formats. NCD measures the distance based on the similarity of two malware's binary sequences. The higher the similarity between the two binary sequences, the smaller the NCD [6, 9]. Specimens of the same IoT malware family are often created based on the same source code with minor modifications [2]. Therefore, these specimens have high binary similarities and can be expected to be divided into close clusters.

A typical method to create a phylogenetic tree is the neighbor-joining method [33]. The neighbor-joining method takes a distance matrix over all the objects (malware binaries for our case) as an input and outputs a phylogenetic tree with the tree distance approximating the input distance matrix. However, the neighbor-joining method requires calculating distances of all the pairs of objects, which means $(N^2 + N)/2$ times of compression attempts in our case. Here, $N$ is the number of objects. The computation time of the NCD matrix is problematic when $N$ is large. Therefore, We proposed a scalable method to construct a phylogenetic tree [13, 14]. Then, we execute a clustering algorithm based on the phylogenetic tree.

In our phylogenetic tree construction method, by calculating only a tiny part of the distance matrix, we achieved good scalability while maintaining the accuracy of clustering. Instead of creating large phylogenetic trees all at once, our basic idea was to create small phylogenetic trees and combine them into one big phylogenetic tree. Using the abovementioned algorithm, we can create a phylogenetic tree with far fewer NCD computations than $(N^2 + N)/2$ times. Our experiments using 65,494 IoT malware specimens show that our fast algorithm reduced the computational cost by 97.52 % compared to the neighbor-joining method.

Clustering is done by appropriately dividing the phylogenetic tree into subtrees. In our work [13], we first applied the `Inconsistency Coefficient` [3] as the division criterion. It worked well

[1] https://www.virustotal.com

on a phylogenetic tree consisting of 4,109 ARM-architecture IoT malware. But when we applied the `Inconsistency Coefficient` to a much larger, multi-architecture malware set, the clustering accuracy dropped significantly. Hence here, We applied the minimum description length (MDL) criterion as the division criterion [32]. The `MDL Criterion` decides whether to divide a cluster (subtree) into smaller clusters. We used a large-scale dataset containing 65,494 malware specimens to evaluate our method's scalability. By improving the clustering algorithm using `MDL Criterion`, our clustering algorithm achieved significantly higher accuracy than the `Inconsistency Coefficient`. In the best case, the family name clustering accuracy was 95.5% and the architecture name accuracy was 99.3%.

Furthermore, considering that new specimens are added to the dataset daily or weekly, reconstructing the phylogenetic tree every time new specimens are collected is time-consuming and resource-intensive. Therefore, we also proposed an online processing algorithm that directly adds new specimens to the clusters without fully reconstructing the phylogenetic tree or clustering it again. Our experiments using 65,494 IoT malware specimens show that the online processing algorithm reduced 33% of the computational cost than the batch processing algorithm while maintaining a clustering accuracy of 94%.

We also attempted to furthermore improve the phylogenetic tree construction algorithm by designing a new seed set selection method. The seed set is like the core of our phylogenetic tree. It is randomly selected so that the seed set can represent the whole dataset. We tried to design a smarter method of seed set selection. Until now, we attempted many new methods and among them, only one method achieved the same level of clustering accuracy with a much smaller variance. Although this result is far from what we expected, we showed the possibility and an idea to improve the clustering accuracy by proposing a new seed set selection method.

Besides the phylogenetic tree construction, we have a different approach for malware clustering by applying active learning. In our phylogenetic tree construction algorithm, the distance matrix is partly calculated based on the randomly selected seed set. In this research, we applied active learning to decide where of the distance matrix to calculate [15]. The active clustering algorithm has been proposed in [16] and [45]. In this research, we implementatd [45]'s method and evaluated it with 3,008 IoT malware. Our experiment showed that the active clustering algorithm observed about 20% lesser NCD than random sampling. However, the runtime of the combinatorial optimization algorithm is too long, even if we change the implementation language to a much faster one "Julia". Therefore, we showed that the active clustering algorithm is effective but currently the run-time problem makes it not very scalable.

# Chapter 2

# Scalable and Fast Algorithm for Constructing a Phylogenetic Tree

In this chapter, we introduce the Scalable and Fast Algorithm for Constructing a Phylogenetic Tree that we proposed in [13]

## 2.1 Preliminaries

In this section, we first introduce two existing methods we applied in our study.

### 2.1.1 Normalized compression distance

In this study, we used NCD to measure the similarity (i.e., distance) between IoT malware binaries. NCD is an information-theoretic measure of the similarity between two objects [22]. The similarity is calculated based on the compression rate of the objects. NCD measures the similarity of objects regardless of their format or structure, e.g., documents, pictures, programs, music, etc.

For two objects $x$ and $y$, NCD is defined as follows:

$$\mathrm{NCD}(x,y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}},$$

where $C(x)$ is the length of $x$ that is compressed by compression program $C$, and $xy$ is the concatenation of objects $x$ and $y$. When compressing $xy$, the compression program will first compress $x$ and then use the information of $x$ to compress $y$. Thus, the more similar $x$ and $y$ are, the higher is the compression rate of $xy$. This means that the value of $C(xy)$ will be similar to that of $C(x)$ or $C(y)$ and result in the NCD value close to zero.

### 2.1.2 Neighbor-joining method

The neighbor-joining method [33] is an algorithm for creating a phylogenetic tree $T$ using a distance matrix $d$ defined on a finite set $L$ as input. Every leaf of the phylogenetic tree represents an object of $L$. The tree distance between two nodes of $T$ is defined by the total branch length of the path between the two nodes. The tree distance is an approximation of the input distance matrix $d$.

In the neighbor-joining method, two nodes $i$ and $j$ that minimize $S_{ij}$, defined below, join a newly created node $p$.

$$S_{ij} = \frac{1}{2(N-2)} \sum_{k \in L \setminus \{i,j\}} (d_{ik} + d_{jk}) + \frac{1}{2} d_{ij} + \frac{1}{N-2} \sum_{a,b \in L \setminus \{i,j\}} d_{ab},$$

where $d_{ij}$ is the $(i, j)$ entry of the given distance matrix $d$. Nodes $i$ and $j$ are deleted from the set $L$, and node $p$ is added instead. This step is repeated until all the nodes are linked. The pseudocode of the neighbor-joining method is shown in Algorithm 1, where $D$ is the set of branch lengths.

**Algorithm 1** Neighbor-joining method

**Require:** finite dataset $L$ with a distance matrix $(d_{ij})$ over $L \times L$
**Ensure:** phylogenetic tree $T$
1: **while** $|L| \geq 2$ **do**
2: choose $(u, v)$ $(u \neq v)$ which minimizes $S_{u,v}$
3: create a new node $p$, $V := V \cup \{u, v, p\}$, $E = E \cup \{\{u, p\}, \{v, p\}\}$
4: branch length of$\{u, p\}$ is defined as:
$D_{u,p} = (d_{u,v} + \frac{\sum_{k \in L \setminus \{u,v\}} (d_{uk} - d_{vk})}{|L|-2})/2$, $L := (L \setminus \{u, v\}) \cup \{p\}$
5: branch length of $\{v, p\}$ is defined as:
$D_{v,p} = (d_{u,v} + \frac{\sum_{k \in L \setminus \{u,v\}} (d_{vk} - d_{uk})}{|L|-2})/2$
6: for $w \in L \setminus \{p\}$, $d(w, p) := (1/2)(d_{u,w} + d_{v,w} - d_{u,v})$
7: **end while**
8: return $T = (V, E), D$

## 2.2 Related Works

### 2.2.1 binary-level static analysis

We applied NCD for feature extraction that directly calculates the similarities between malware's byte sequences. Other binary-level feature extraction methods include N-grams [20, 30, 44], entropy-based [12, 28], and image-based techniques [5, 7, 24], which convert a file's entire sequence of bytes into a picture, where each byte represents the grayscale of a pixel.

Compared to N-gram, N-gram lacks long-term dependence, because it only considers the N-1 bytes before the current byte. Moreover, in practice, N is usually ranged from 2 to 8 due to the computation complexity [20]. On the contrary, NCD's feature extraction is based on the compression algorithm. In our method, we chose Lempel–Ziv–Markov chain algorithm (LZMA) as the compression algorithm. Markov chain algorithm has much longer-term dependence than N-gram so the Markov chain algorithm can match more same patterns in the byte sequences. Furthermore, N-gram can only use N-byte words as the dictionary, but LZMA has a more flexible dictionary.

In the entropy-based feature extraction, information in the byte sequences will lose while converting the byte sequences into entropy. Different byte sequences can result in the same entropy value.

The image-based method converts byte sequences to an image. Each byte represents the grey scale [0-255] of a pixel. The image can later be used as input for Neural Networks or other methods. The advantage of this method is its robustness [1]. Our method and N-gram match

Table 2.1: Summary of the studies constructing phylogenetic tree for malware clustering

| Reference | Malware analysis | | Phylogenetic tree | | Scalability |
|---|---|---|---|---|---|
| | Analysis method | Feature extraction | Size | construction method | |
| Karim et al. 2005 [21] | Static analysis | N-gram and N-perm | 170 | UPGMA | No |
| Vinod et al. 2012 [41] | Static analysis | NCD | 790 | Not mentioned | No |
| Hsiao et al. 2016 [18] | Dynamic analysis | system call | 1,200 | unweighted pair group | No |
| Wehner 2007 [40] | Static analysis | IDA | 1,200 | neighbor-joining | No |
| Bailey et al. 2007 [3] | Dynamic analysis | Behaviors | 3,700 | The shortest distance | No |
| Cozzi et al. 2020 [10] | Both dynamic and static analysis | Code-based | 36,574 | HNSW + the minimum spanning tree | Yes |

TABLE 2.2: Summary of the studies about large-scale malware clustering

| Reference | Malware analysis | | Size | Clustering method | Scalability |
|---|---|---|---|---|---|
| | Analysis method | Feature extraction | | | |
| Dam et al. 2021 [11] | Static analysis | System call graph | 21,000 | graph clustering | Yes |
| Ricck et al. 2011 [31] | Dynamic analysis | Behavior | 33,698 | Prototype-based clustering | Yes |
| Torabi et al. 2021 [39] | Static analysis | Strings-base | 49,272 | ClusterONE | Yes |
| Bayer et al. 2009 [4] | Dynamic analysis | System call and control flow | 75,692 | LSH | Yes |
| Hu et al. 2013 [19] | Dynamic analysis | Code instruction | 130,000 | Prototype-based clustering | Yes |
| Oliver et al. 2020 [26] | Static analysis | 3-grams and TLSH | 10 million | HAC-T | Yes |

exactly the same pattern between byte sequences, while Neural Networks can deal with minor alterations.

**phylogenetic tree for malware clustering**

Many studies have applied the phylogenetic tree to cluster malware samples and help the analyst to better understand the evolutionary relationships between malware. But almost all of these studies analyzed small malware sets. Because their methods require the computation of the whole distance matrix, the $O(N^2)$ complexity makes it impossible to apply these methods to a large malware set. A summary table of related studies constructing a phylogenetic tree is shown in Table 2.1.

Karim et al. [21] proposed a method to calculate the distance between all malware pairs using n-perms and created a phylogenetic tree. Vinod et al. [41] also used NCD to calculate the distance between 790 malware samples. They did not mention their method for constructing the phylogenetic tree, but they noted that they calculated all pairs of malware's distance. In [18], Hsiao et al. extracted malware features based on dynamic analysis, and the unweighted pair group method with arithmetic mean was used to construct their phylogenetic tree. The size of the dataset was 1,200. Wehner [40] used IDA, a binary code analysis tool for reverse engineering, to transfer executable malware to assembly code and calculated the similarity based on it. They used the neighbor-joining method to construct the phylogenetic tree. Their experiment contained 1,200 malware specimens. Bailey et al. [3] proposed a representative automatic clustering method based on the dynamic analysis of malware. Using the dynamic analysis log of 3,700 Windows malware samples as input, the NCD matrix between the operation log data of all the samples was calculated, and then a phylogenetic tree was created using the shortest distance method. In addition, they proposed a clustering criterion called `Inconsistency Coefficient` and performed hierarchical clustering based on it. We also used it in our studies. However, there was a problem in that the clustering accuracy dropped significantly for a large-scale, multi-architecture phylogenetic tree. Therefore, in this study, we improved the clustering method and performed clustering using the `MDL Criterion`. All these studies calculated all pairs of malware distance, because any $O(N^2)$ algorithm does not scale well, they are not suitable for constructing large-scale phylogenetic trees.

To the best of our knowledge, [10] is the only related study on the construction of large-scale phylogenetic trees. Cozzi et al. separately constructed phylogenetic trees for different architectures, and the largest one contains 36 thousand malware samples. They used hierarchical navigable small world graphs (HNSW) to reduce the distance computation significantly. Compared to their study, our phylogenetic tree almost doubles their size.

**large-scale malware clustering**

Although there is only one related study about large-scale phylogenetic tree construction, there exist several approaches to reduce the computational cost of large-scale malware clustering. A summary table of large-scale malware clustering is shown in Table 2.2.

Bayer et al. [4] proposed a large-scale clustering method based on dynamic analysis. They used the malware dynamic analysis log as input for clustering and performed clustering by calculating approximately 2% of the distance matrix without calculating the distance between all data pairs using the locality sensitive hash (LSH). However, like Bailey et al.'s method, dynamic analysis requires several minutes to run each malware specimen. By contrast, our static analysis method does not require the malware to be executed.

Olivr et al. [26] proposed a clustering method based on a static analysis that uses trend locality sensitive hashing (TLSH) to transfer every malware binary into fixed-length hash digests. They then clustered all digests into clusters using a clustering method called HAC-T within $O(N \log N)$ time. Torabi et al. [39] proposed a strings-based method to analyse and cluster 49,272 IoT malware. They extracted useful strings and calculated its similarities using Jaccard and overlap similarity coefficients. Moreover, they applied ClusterONE [42] algorithm to perform clustering analysis. Ricck et al. [31] and Hu et al. [19] took a different approach: a prototype-based clustering algorithm that reduces runtime complexity by performing clustering only on representative samples (prototypes). The remaining malware specimens are associated with their closest prototype in the feature space. Dam et al. [11] also took a very different approach: they use IDA pro to extract the system call graphs for each malware and transfer the graphs into vectors applying LSTM.

The neighbor-joining method requires a complete distance matrix to construct a phylogenetic tree. The $O(N^2)$ computational cost is a problem when the dataset is large. Therefore, we proposed a fast and scalable algorithm that only needs to calculate a small part of the distance matrix to construct a phylogenetic tree. Our algorithm is an improvement of the algorithm for estimating a phylogenetic tree of big data proposed in [43].

### 2.2.2 The Algorithm for Estimating a Phylogenetic Tree of Big Data

First, the algorithm randomly selects a seeds set $S \subset L$ $(|S| = k, |L| = N)$ with $k$ $(k << N)$. Then, it calculates the distance matrix between $S$ and $L$, and, using the distance matrix over $S \times S$, creates a phylogenetic tree $T$ using the neighbor-joining method. For each element $z$ in $L \setminus S$, using the distance matrix over $(L \setminus S) \times S$, it inserts $z$ into the appropriate edge $l$ that minimums a cost function $W_{T@l}$. We will not discuss the details of $W_{T@l}$ in this thesis so please refer [8] for more details. Name $Z(l)$ as the data set be inserted into $l$, then if $|Z(l)| < h$, it calculates the distance matrix over $Z(l) \cup \{l\}$ and creates a phylogenetic tree $T_Z(l)$ with it using the neighbor-joining method. It then combines $T_Z(l)$ and $T$. Here, $h$ is a predefined threshold. $l$ is included in the recomputation to know which part of $T_Z(l)$ corresponds to $T$. If $|Z(l)| > h$, then it recursively uses this algorithm for $Z(l) \cup l$. The pseudocode of the algorithm is shown in Algorithm 2,

The computation order of calculating $W_{T@l}$ is $O(|S|^4 * |L|)$, it's very time-consuming when $|S|$ is large. Therefore, we improved this part of the algorithm. Instead of inserting data in $L \setminus S$ into the edges of $T$, we link the data to leaves of $T$.

### 2.2.3 Phylogenetic Tree Construction Algorithm

The improved algorithm is outlined as follows. The schematic diagram and the flow chart are shown in Fig. 2.2 and Fig. 2.3, respectively. First, the algorithm randomly selects a seeds set $S \subset L$ $(|S| = k, |L| = N)$ with $k$ $(k << N)$. Then, it calculates the distance matrix between $S$ and $L$, and, using the distance matrix over $S \times S$, creates a phylogenetic tree $T$ using the neighbor-joining method. For each element $z$ in $L \setminus S$, using the distance matrix over $(L \setminus S) \times S$, it links it with the leaf $e$ of $T$, which is nearest to the element $z$.

**Algorithm 2** The Algorithm for Estimating a Phylogenetic Tree of Big Data

**Require:** finite dataset $L$, size $k$ of seeds set, threshold $h$
**Ensure:** phylogenetic tree $T$
choose a certain seeds set $S \subset L$ with $|S| = k$
calculate the distances $d_{ij}$ for $(i, j) \in S \times L$
create a phylogenetic tree $T$ for $S$ by the Neighbor-joining method using $d_{ij}$
**for** $l \in \partial T$ **do**
    $Z(l) = \emptyset$
**end for**
**for** $z \in L \backslash S$ **do**
    $Z(l) = Z(l) \cup \{z\}$, where $l$ minimums $W_{T@l}$
**end for**
**for** $e \in \partial T$ **do**
    **if** $|Z(e)| > h$ **then**
        recursively use Algorithm 2 for $Z(l) \cup \{l\}$
    **end if**
    **if** $1 < |Z(l)| < h$ **then**
        calculate $d_{ij}$ for $(i, j) \in (Z(l) \cup \{l\})^2$ and create a phylogenetic tree $T_Z(l)$ with it.
        replace the corresponding parts of $T$ with $T_Z(l)$
    **end if**
**end for**

To increase the approximation accuracy of the tree distance between set $L \setminus S$, it recalculates the tree distance recursively for each $Z(e)$. If $|Z(e)| < h$, it calculates the distance matrix over $Z(e) \cup \{e\}$ and creates a phylogenetic tree $T_Z(e)$ with it. It then combines $T_Z(e)$ and $T$. Here, $h$ is a predefined threshold. $e$ is included in the recomputation to know which part of $T_Z(e)$ corresponds to $T$. If $|Z(e)| > h$, then it recursively uses this algorithm for $Z(e) \cup e$. The pseudocode of the phylogenetic tree construction algorithm is shown in Algorithm 3, where $\partial T$ denotes a set of all leaves of $T$.

**Algorithm 3** Fast Algorithm for Constructing a Phylogenetic Tree

**Require:** finite dataset $L$, size $k$ of seeds set, threshold $h$
**Ensure:** phylogenetic tree $T$
choose a certain seeds set $S \subset L$ with $|S| = k$
calculate the distances $d_{ij}$ for $(i, j) \in S \times L$
create a phylogenetic tree $T$ for $S$ by the Neighbor-joining method using $d_{ij}$
**for** $e \in \partial T$ **do**
    $Z(e) = \emptyset$
**end for**
**for** $z \in L \backslash S$ **do**
    $Z(e) = Z(e) \cup \{z\}$ where $e$ is nearest to $z$
**end for**
**for** $e \in \partial T$ **do**
    **if** $|Z(e)| > h$ **then**
        recursively use Algorithm 3 for $Z(e) \cup \{e\}$
    **end if**
    **if** $1 < |Z(e)| < h$ **then**
        calculate $d_{ij}$ for $(i, j) \in (Z(e) \cup \{e\})^2$ and create a phylogenetic tree $T_Z(e)$ with it.
        replace the corresponding parts of $T$ with $T_Z(e)$
    **end if**
**end for**

Instead of calculating all pairs of distances, only the colored parts in Fig. 2.2 are calculated in Algorithm 3. As a randomized algorithm, Algorithm 3's computational cost is $O(N^2)$ in the worst case. But in usual cases, the computational cost is reduced from $O(N^2)$ to $O(N \log N)$.

As a randomize algorithm, In the worst case, the time complexity of our algorithm is O($N^2$) and the computation reduction rate is 0%. But in most case, the time complexity of our algorithm is O($N \log N$) and the computation reduction rate is over 95%.

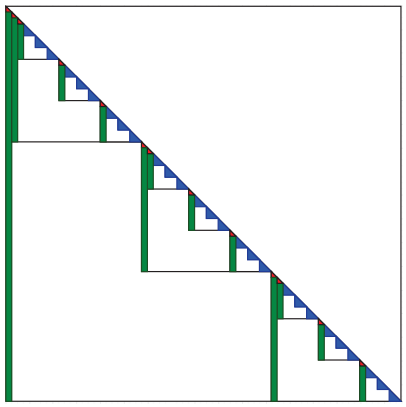


FIGURE 2.1: The schematic diagram of the computation cost of our algorithm

Let $N$ be the size of dataset $L$, $k$ be the size of seeds set $S$ and $h$ be the threshold. The worst case is that all data in set $L\backslash S$ are assigned to a single subset, for example $Z(e_0)$, and other subsets $Z(e_1)$ to $Z(e_{k-1})$ remain empty. And if this happens every time we recursively applying our algorithm for subset $Z(e_0)$, our algorithm will finally calculate the whole distance matrix.

However, in usual case, these $k$ data randomly picked from the dataset can represent the whole dataset to some extent. for the convenience of computation, we assume that data in set $L\backslash S$ are splited into $Z(e_0)$ to $Z(e_{k-1})$ equally. $N$ data are recursive divided into $k$ subsets until the size of the subset is smaller than $h$, so the recursive calculations will totally be $\log_k^{\frac{N}{h}} - 1$ times. The computation cost of seed set is represented by the red parts in Fig.2.1, and it is calculated as:

$$\frac{1}{2}k^2(1+k+k^2+...+k^{\log_k^{\frac{N}{h}}-1}) = \frac{1}{2}k^2 * \frac{h-N}{h(1-k)} = O(N).$$

The computation cost for assigning the $L\backslash S$ is represented by the green parts, and it is calculated as:

$$\begin{aligned} k(N-k+N-k^2+...+N-k^{\log_k^{\frac{N}{h}}}) &= kN\log_k^{\frac{N}{h}} - k^2\frac{h-N}{h(1-k)} \\ &= O(N\log N). \end{aligned}$$

Finally, the computation cost of neighbor-joining method is represented by the blue parts, and it is calculated as:

$$\frac{1}{2}h^2 * k^{\log_k^{\frac{N}{h}}} = \frac{1}{2}hN = O(N).$$

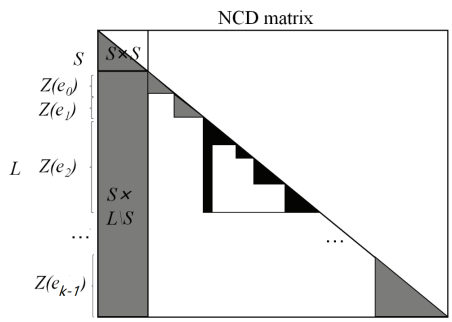



FIGURE 2.2: The schematic diagram of the phylogenetic tree construction algorithm

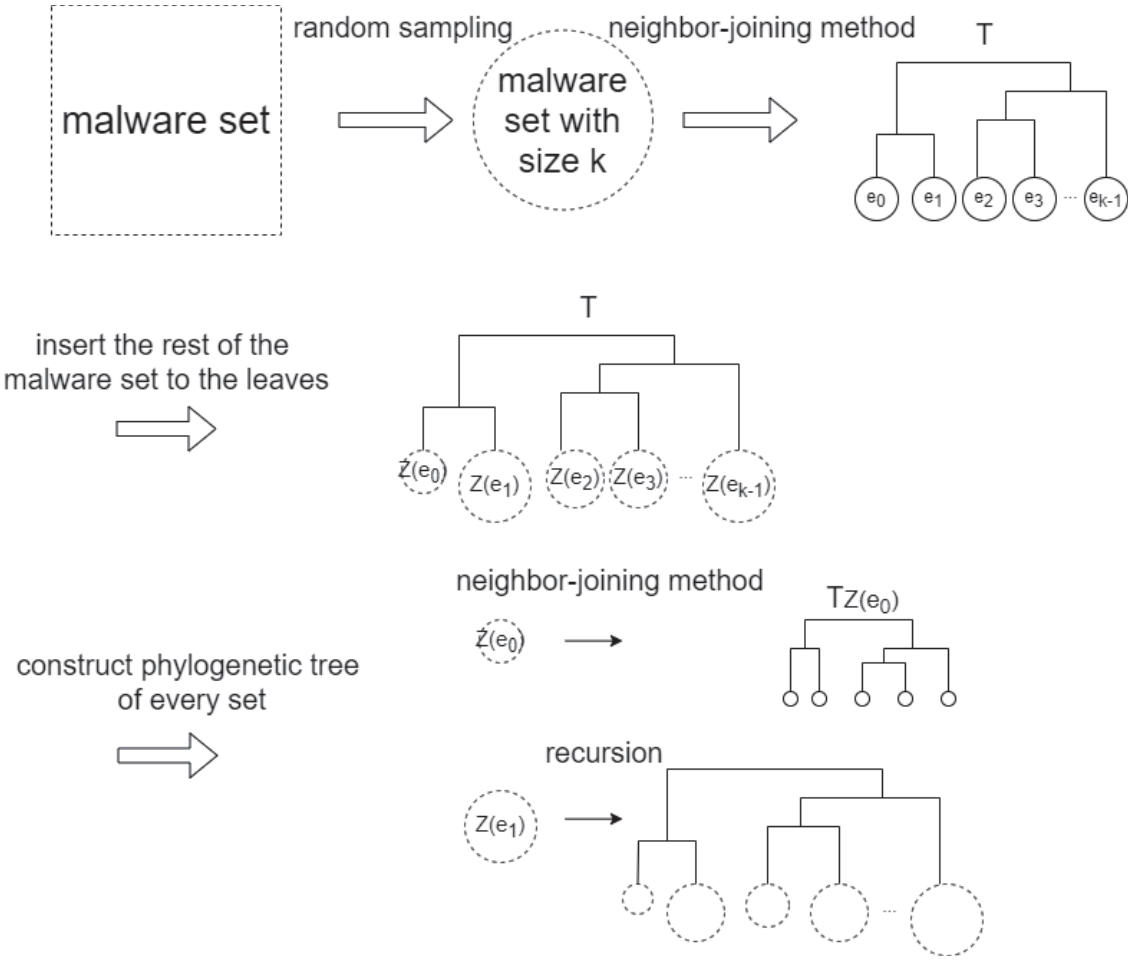



FIGURE 2.3: The flow chart of the the phylogenetic tree construction algorithm

# Chapter 3

# Clustering Algorithm of Phylogenetic Tree with Application to IoT Malware Clustering

Since the burden on the analyst cannot be reduced by simply constructing the phylogenetic tree, it is necessary to divide the phylogenetic tree into appropriate clusters. In this chapter, we introduce our clustering algorithm and the experiments.

## 3.1 Clustering Algorithm

In the early phase of our study, we achieved a good clustering accuracy by using the clustering method based on the `Inconsistency Coefficient` [3], but when it was applied to a larger, multi-architecture dataset, performance decreased considerably. To solve this problem, we changed the clustering method to `MDL Criterion` [32].

The `MDL Criterion` is a model selection criterion based on information theory [32]. `MDL Criterion` is defined as below:

$$\mathrm{MDL} = -2\log L(\hat{\theta}; x) + m_j \log(n). \tag{3.1}$$

where $L$ is the likelihood function, $\theta$ is the maximum likelihood estimate, $m_j$ is the number of dimensions of $\theta$, n is the number of data samples. The description length includes the description length of the model and the description length of the data when the model is given. When using a complicated model, the data description length can be short, but the description length of the model becomes long. When using a simple model, the description length of the model is short, but the description length of the data becomes long. An appropriate model should be selected by balancing both and minimizing the total description length.

The outline of the clustering algorithm is shown below. First, given one cluster C, it is assumed that the data $x$ contained in the cluster follows a normal distribution, and the description length (DL) of the cluster is calculated by the following.

$$\begin{aligned}\mathrm{DL}(C) &= -2\log L(\hat{\theta}; x) + m_j \log(n) \\ &= n\log\Big(\frac{1}{n}\sum_{i=1}^{n}|x_i - \bar{x}|^2\Big) + m_j \log n + n(\log 2\pi + 1).\end{aligned}$$

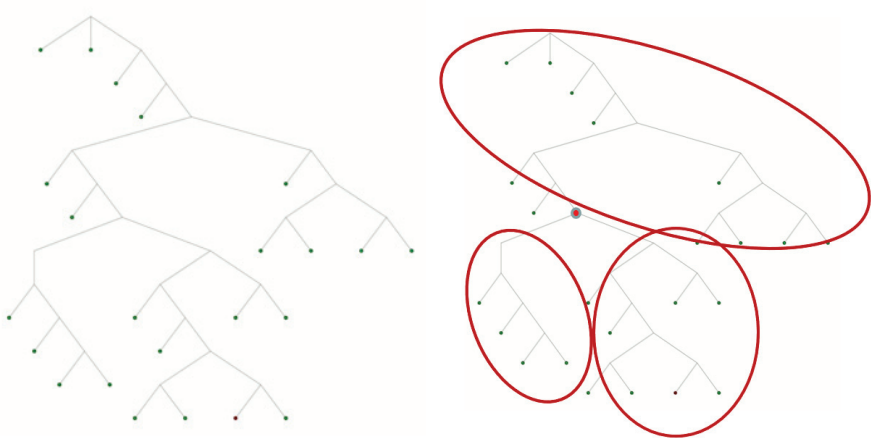
Here, $L$ is the likelihood function, $\theta$ is the maximum likelihood estimate, $n$ is the number of data samples in the cluster, and $m_j$ is the number of dimensions of the parameter.

In our model [14], this equation cannot be directly calculated because it is originally designed for Euclidean space. We must approximate it as follows: $x_i - \bar{x}$ is calculated using the tree distance between the malware $i$ and the central malware $\bar{x}$. The central malware is defined as the malware that has the smallest sum of squares of the column elements in the cluster's tree distance matrix. $m_j$ becomes a free parameter that determines the fineness of the division of the phylogenetic tree. Note that different information criteria like Akaike information criterion (AIC) or Bayesian information criterion (BIC), or `MDL Criterion` are only different at the weight of $m_j$, so since we regard $m_j$ as a free parameter and use parameter tuning to choose the best $m_j$, apply any information criterion is actually the same.

The phylogenetic tree is a 3-regular graph (which means every node has three neighbors); therefore, when dividing a cluster (subtree), it will be divided into three subtrees, viz., $C_1, C_2, and\ C_3$. We divide a cluster at its central node, which is defined as the node which has the smallest sum of the squares of the column elements in the tree distance matrix. A schematic diagram of the division example is shown in Figure 3.1 The description length of the divided cluster is calculated by:

$$\begin{aligned}\mathrm{DL}(C_1, C_2, C_3) = &\sum_{k=1}^{3} n_k \log\Big(\frac{1}{n_k}\sum_{i=1}^{n} |x_{ki} - \bar{x_k}|^2\Big) \\ &+ 3m_j \log n + n(\log 2\pi + 1).\end{aligned}$$

Here, $x_k$ is the malware sample contained in the cluster $C_k$, and $n_k$ is its number. If $\mathrm{DL}(C_1, C_2, C_3) >$



(a) before dividing (b) after dividing

FIGURE 3.1: Example of dividing a cluster

$\mathrm{DL}(C)$, the cluster before the division is considered better, and the division is rejected. If $\mathrm{DL}(C_1, C_2, C_3) <$ $\mathrm{DL}(C)$, the cluster after division is considered to be better, and the cluster will be divided into three clusters. The determination of division will continue for each small cluster.

While dividing the phylogenetic tree, many small clusters are formed. After completing the division, we merge these small clusters into their respective close clusters. The distance between clusters is defined by the tree distance between their central nodes. The merging process was conducted targeting the clusters containing one hundred or fewer specimens with other clusters in the order of closer distance. The merging is only done when the description length $\mathrm{DL}(C)$ of the cluster after merging is smaller than the description length $\mathrm{DL}(C_1, C_2)$ of the clusters before merging. The pseudocode of the algorithm is shown in Algorithm 4.

**Algorithm 4** Algorithm for Clustering a Phylogenetic Tree

**Require:** Phylogenetic tree $T$
**Ensure:** Clusters $Tc$

```
Function Cluster(C)
find the central node of C.
divide the cluster C into C_1, C_2, C_3 at the central node.
if DL(C_1, C_2, C_3) > DL(C) then
    Tc = Tc ∪ C
end if
if DL(C_1, C_2, C_3) < DL(C) then
    Cluster(C_1)
    Cluster(C_2)
    Cluster(C_3)
end if
EndFunction
Tc = ∅
Tc = Cluster(T)
for T_i ∈ Tc do
    if |T_i| < 100 then
        Tn= T_i's nearest cluster
        C= merge T_i and Tn
        while count < 10 do
            if DL(T_i, Tn) > DL(C) then
                Tc= Tc − T_i − Tn
                Tc = Tc ∪ C
                break
            else
                Tn=T_i's next nearest cluster
                count++1
            end if
        end while
    end if
end for
```

## 3.2 Experiment and Results

In this section, we evaluate our algorithm with 65,494 IoT malware. In particular, it shows 1) how well our fast algorithm reduces the computational cost compared with the neighbor-joining method, 2) how much the `MDL Criterion` improves the clustering accuracy compared with `Inconsistency Coefficient`.

Table 3.1: Breakdown of ISA and malware family names of the dataset

| ISA | Malware Family | | | | | | Total |
|---|---|---|---|---|---|---|---|
| | Bashlite | Mirai | Tsunami | mobidash | Wroba | others | |
| ARM | 7031 | 11492 | 298 | 360 | 301 | 2510 | 21992 |
| MIPS | 4918 | 6321 | 258 | 78 | 263 | 329 | 12167 |
| Intel | 4644 | 4971 | 720 | 597 | 237 | 1808 | 12977 |
| x86 | 1753 | 753 | 141 | 180 | 59 | 697 | 3583 |
| PowerPC | 1913 | 2983 | 63 | | | 11 | 4970 |
| Renesas | 947 | 1907 | 49 | | | 5 | 2908 |
| SPARC | 1681 | 2226 | 26 | | | 4 | 3937 |
| Motorola | 929 | 1900 | 33 | | | 5 | 2867 |
| others | 16 | 51 | | 2 | | 24 | 93 |
| Total | 23832 | 32604 | 1588 | 1217 | 860 | 5393 | 65494 |

### 3.2.1 Dataset

We collected 65,494 Linux malware specimens (mostly IoT malware) from VirusTotal, from November 2018 to February 2019. The breakdown of the dataset is shown in Table 3.1. The malware family name was decided by AVClass [36]. AVClass is implemented as a Python tool to label malware samples using VirusTotal JSON reports as input.

### 3.2.2 Experimental Setup

For performance comparison, we constructed phylogenetic trees with both the our fast algorithm and the neighbor-joining method. We then clustered them with both the `MDL Criterion` and the `Inconsistency Coefficient`, which is the method we applied in the early phase. To evaluate the clustering performance, we used 10-fold cross-validation. After clustering 90% of the specimens, we assigned the remaining 10% to their nearest cluster to calculate the accuracy.

Moreover, to investigate the impact of dataset size on computational cost reduction and clustering accuracy, the experiment included two parts: First, we evaluated our algorithm with a small number of specimens: 4,000 specimens, which are randomly selected from the malware set. Second, we evaluated our algorithm with 65,494 specimens.

The size $k$ of the Seed set $S$ was set to 1% of the number of specimens. The recursive computation threshold $h$ was set to 5000. The clustering parameter $m_j$ was chosen from 4, 5, 7, 10, 15, 20, 25, 30.

We implemented our algorithm using `R`. The compression program `xz` command (version 5.1.0 alpha) of Linux was used to compress malware binaries for computing NCDs. `xz` adopted the Lempel-Ziv-Markov chain algorithm (LZMA) [35] for compression.

As a benchmark for our algorithm, we used a conventional scheme: it first compressed every pair of 65,494 malware binaries to compute the NCDs between them, and then constructed the phylogenetic tree of the malware with the neighbor-joining method described in subsection 2.1.2. Because of the large size of the dataset, using R's library to run the neighbor-joining method was

impossible; therefore, we used the library available for the textttJulia programming language as an alternative [38].

The experiment was conducted on a 2.6 GHz Intel-Xenon-Gold-6126 CPU. In our algorithm and the conventional scheme, the compression of NCD was computed in parallel with 80 threads.

### 3.2.3 Evaluation Metrics

1. ***RCR***: To measure how well the compression attempts are reduced by our fast algorithm, we define the rate of compression-attempt reduction RCR as follows:

$$\text{RCR} = (1 - \frac{\text{\# of compression attempts by ours}}{(N^2+N)/2}) \times 100\%$$

   where $N$ is the number of specimens, which equals 65,494. The RCR decreases as the compression attempts are further reduced by our algorithm.

2. ***clustering accuracy***: The clustering accuracy was evaluated by a 10-fold cross-validation. Specifically, the malware set was divided into ten parts, and clusters were created using 90% of the specimens. Each of the remaining 10% test specimens was assigned to their nearest cluster. If the family name of the cluster and the family name of the test specimen is the same, then the test specimen is correctly clustered. The architecture name accuracy is calculated in the same way. The distance between a specimen and cluster is defined by the NCD between the specimen and the cluster's center specimen. The cluster's family name is decided by the majority specimen's family name in that cluster.

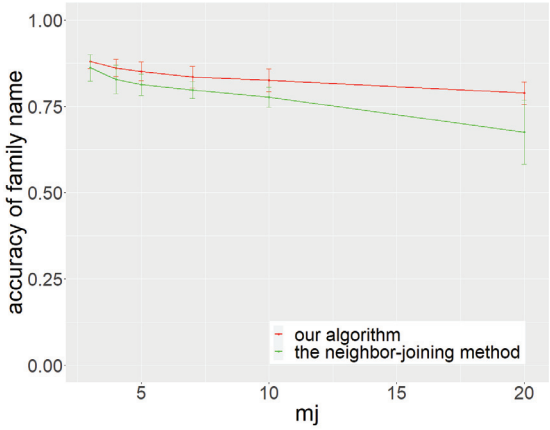
### 3.2.4 Evaluation results with 4,000 Specimens

This experiment is designed to investigate the impact of dataset size on RCR and clustering accuracy, which will be introduced in next subsubsection. In the experiment using 4,000 specimens, the RCR was 94.01%. The clustering accuracy is shown in Fig. 3.2(a). The red line represents our algorithm, and the green line represents the neighbor-joining method. After constructing the phylogenetic trees, they were both divided into clusters based on the `MDL Criterion`. In Fig. 3.2(a) and Fig. 3.2(b), the horizontal axis is the parameter $m_j$ introduced in section 3.1 that determines how finely the phylogenetic tree is divided. As $m_j$ becomes smaller, the phylogenetic tree is divided into smaller clusters, whose number increases exponentially, and the clustering accuracy also increases simultaneously. Our algorithm achieved a slightly higher family name clustering accuracy than the neighbor-joining method, especially when $m_j$ is large.

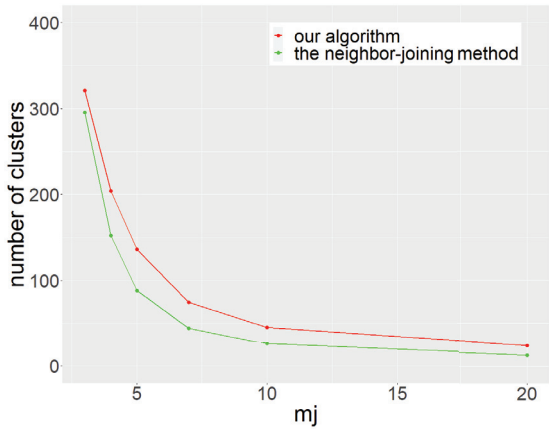
For the same $m_j$, a slightly smaller number of clusters were created using the neighbor-joining method. Because the larger number of clusters, the more choices for test specimens, and the higher clustering accuracy, we changed the horizontal axis variable to the number of clusters in Fig. 3.3(c) and Fig. 3.3(d) to show the difference better. However, our algorithm still achieves a slightly higher accuracy.

Fig. 3.2(c) and Fig. 3.2(d) show the accuracy achieved by our algorithm in clustering family name and architecture name, respectively. For comparison, we used different methods to construct and cluster the phylogenetic tree. The combinations of different methods are:

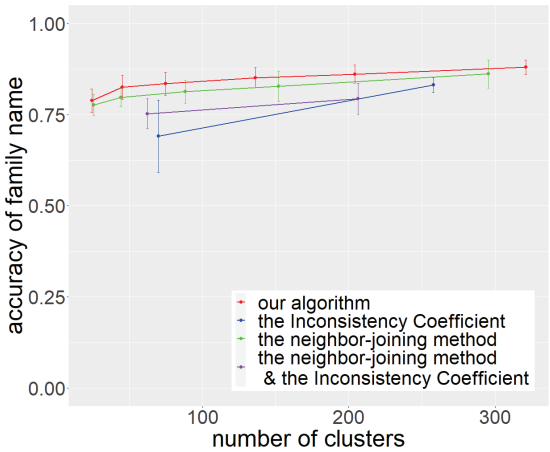
- Our fast algorithm and `MDL Criterion`, represented by the red line.
- Our fast algorithm and the `Inconsistency Coefficient`, represented by the blue line.
- The neighbor-joining method and `MDL Criterion`, represented by the green line.



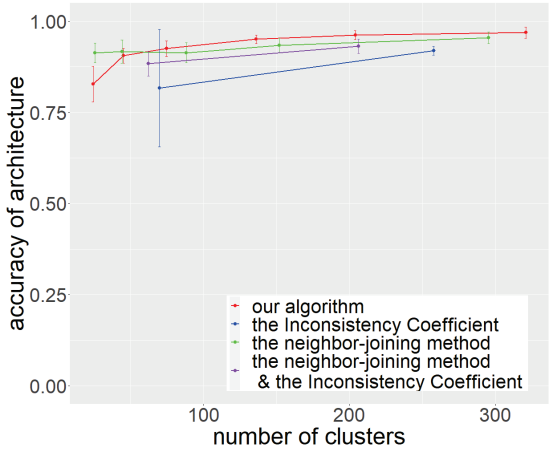

(a) $m_j$-family name accuracy




(b) $m_j$-number of clusters




(c) family name accuracy comparison of 4 methods



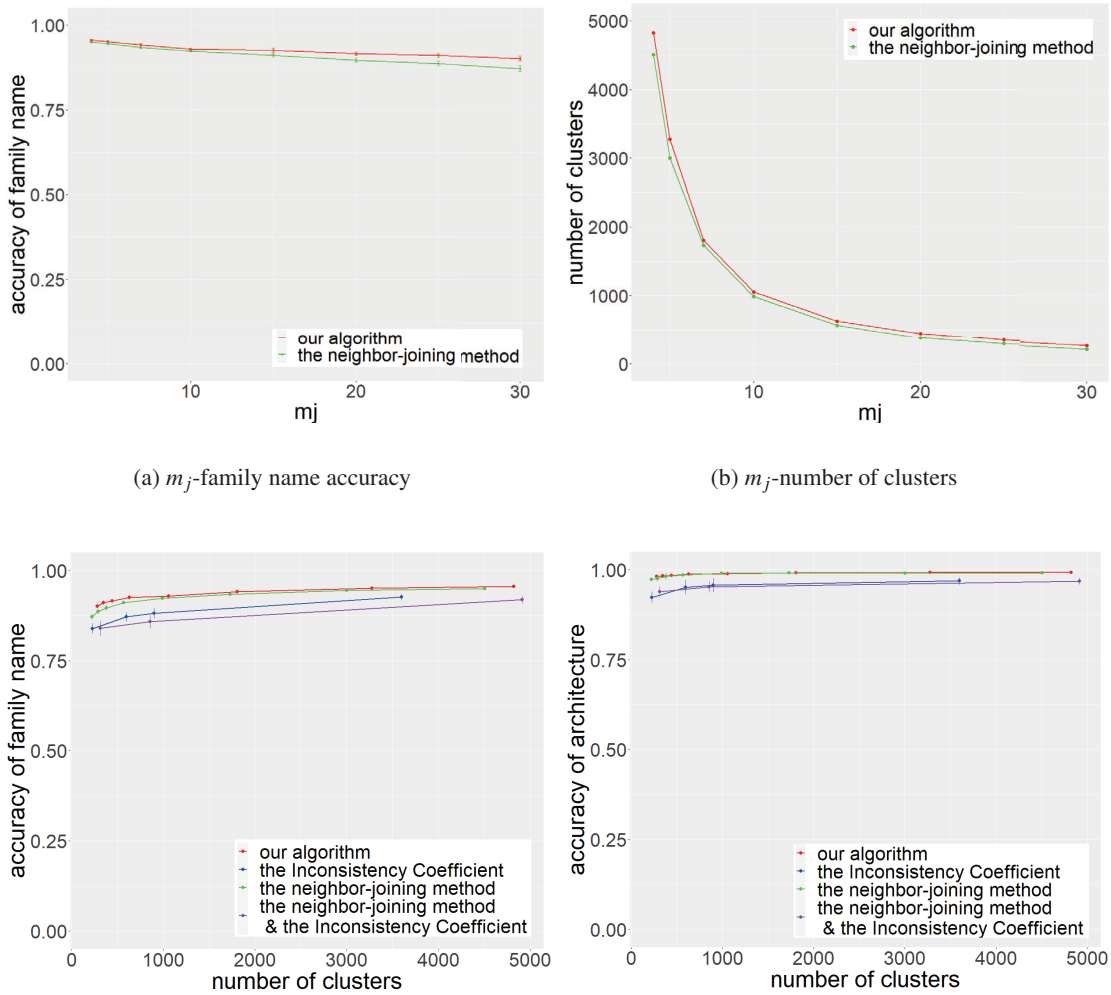

(d) architecture name accuracy comparison of 4 methods

FIGURE 3.2: Experimental results with 4,000 specimens and comparison between our algorithm, the neighbor-joining method, and the `Inconsistency Coefficient`.

- The neighbor-joining method and the `Inconsistency Coefficient`, represented by the purple line.

Our fast algorithm combined with the `MDL Criterion` achieved the best clustering accuracy among the four methods. The results show that our proposed algorithm successfully maintains the clustering accuracies of the neighbor-joining method while significantly reducing the computational cost. Moreover, the clustering algorithm applying the `MDL Criterion` successfully improved the clustering accuracies than the `Inconsistency Coefficient`. The `Inconsistency Coefficient` is a clustering criterion designed by Bailey et al. [3] based on experience. We believe our method achieved a better result because we have a better theoretical basis.

### 3.2.5 Evaluation results with 65,494 Specimens




(a) $m_j$-family name accuracy

(b) $m_j$-number of clusters

(c) family name accuracy comparison of 4 methods

(d) architecture name accuracy comparison of 4 methods

FIGURE 3.3: Experimental result and comparison between our algorithm, the neighbor-joining method and the `Inconsistency Coefficient`.

The phylogenetic tree constructed using 65,494 specimens is shown in Fig. 3.4, where every malware is colored according to its family name. The red points represent *Bashlite*, the green points represent *Mirai*, and other colors represent other families. Our fast algorithm reduced the number of compression attempts by 97.52 % compared with the neighbor-joining method. This

result shows that the larger the dataset, the higher the RCR achieved. This is because our fast algorithm reduces the computational cost from $O(N^2)$ to $O(N \log N)$, and $N/\log N$ increases with $N$. To calculate the input NCD matrix of the neighbor-joining method, $(65,494^2 + 65,494)/2$ pairs of malware specimen are compressed in 80 threads with a 2.6 GHz Intel-Xenon-Gold-6126 CPU, which took 110 days. As for the neighbor-joining algorithm, we used the PhyloNetworks [38], a library of Julia, to construct the phylogenetic tree. The neighbor-joining algorithm was executed in 1 thread and took ten days. On the contrary, our method calculated only 2.48% of the NCD matrix. Including the time for tree construction, it only took three days.

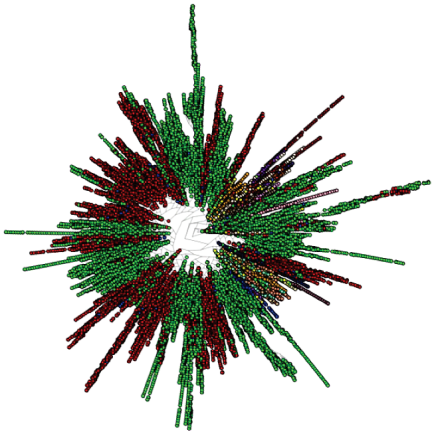


FIGURE 3.4: Phylogenetic tree constructed using our algorithm

It can be seen from Fig. 3.3(c) and Fig. 3.3(d) that our algorithm achieved a slightly higher clustering accuracy than the neighbor-joining method. In addition, it achieved the best accuracy among the four methods. Comparing the red and the blue lines, our clustering algorithm based on `MDL Criterion` significantly improved the family name clustering accuracy and architecture clustering accuracy. For instance, when $m_j = 7$, the phylogenetic tree was divided into 1808 clusters. The family name accuracy of the `MDL Criterion` was 94.1%, and the architecture name accuracy was 99.1%. Compared with the `Inconsistency Coefficient` with the same number of clusters, the family name accuracy was 89.6%, and the architecture name accuracy was 96.1%, which is an improvement of 4.5 percentage points of family name accuracy and 3.0 percentage points of architecture name accuracy. This result also shows that the larger the dataset, the higher the clustering accuracy achieved. We believe this is because the size of the seed set is set to 1% of the size of the dataset. The larger dataset can result in a smaller variation of the seed set, which means the randomly picked seed set can better represent the whole dataset.

We randomly pick one case in the 10-fold cross-validation experiment and show the confusion matrix of the test malware set in Figure 3.5. Although we used an unbalanced dataset, the proportion of the Bashlite and Mirai family groups is over 86%, but our algorithm can cluster minor families successfully. Figure 3.5 shows that we achieve high accuracy for most minor families.

| | Actual family | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mirai | Bash-lite | Tsu-nami | Mobi-dash | Others | Wroba | Dof-loo | Setag | Fake-bank | Ddo-stf | Locker | Xor-ddos | Lot-oor | Ha-jime | Hi-ddad | Dns-amp | Xmrig | Fee-jar | Fifo-reg | ssh-door | Root-nik |
| Predicted family | | | | | | | | | | | | | | | | | | | | | |
| Mirai | 3170 | 118 | 25 | 0 | 80 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Bashlite | 59 | 2249 | 54 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Tsunami | 2 | 16 | 79 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mobidash | 0 | 0 | 0 | 130 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| Others | 2 | 1 | 2 | 0 | 81 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 |
| Wroba | 0 | 0 | 0 | 0 | 3 | 79 | 0 | 0 | 13 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 |
| Dofloo | 1 | 0 | 0 | 0 | 3 | 0 | 77 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Setag | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 62 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fakebank | 0 | 0 | 0 | 0 | 1 | 4 | 0 | 0 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ddostf | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 41 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Locker | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Xorddos | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Lotoor | 1 | 0 | 0 | 0 | 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Hajime | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Hiddad | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 0 | 0 | 0 | 0 | 0 | 0 |
| Dnsamp | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 0 | 0 |
| Xmrig | 2 | 0 | 1 | 0 | 1 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 0 |
| Feejar | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 9 | 0 | 0 | 0 |
| Fiforeg | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| Sshdoor | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 |
| Rootnik | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| accuracy(%) | 97.90 | 94.34 | 48.77 | 100 | 38.03 | 90.80 | 89.53 | 96.88 | 66.67 | 97.62 | 100 | 96.67 | 48 | 95 | 100 | 84.62 | 100 | 100 | 22.22 | 100 | 14.29 |

FIGURE 3.5: Confusing matrix of test malware specimens

# Chapter 4

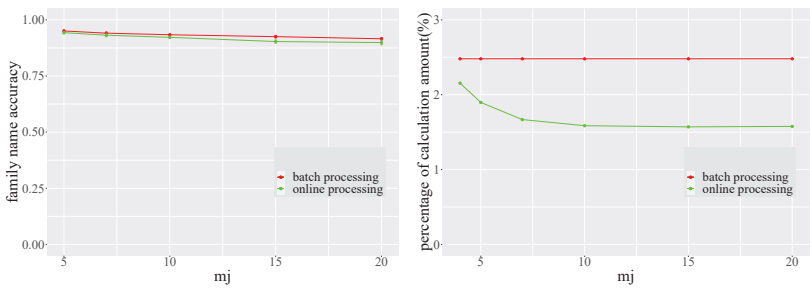
# Online Processing of the Clustering and Current Work in Progress

Since new malware specimens are collected by honeypots every day, reconstructing the phylogenetic tree each time new specimens are added to the dataset will be time-consuming. Therefore, we propose an online processing algorithm that adds new specimens to the existing clusters without the reconstruction of the phylogenetic tree and clustering again. We also introduce our current work in progress in this section.

## 4.1 Online Processing Algorithm

After creating a phylogenetic tree and clustering it, we calculate the distance between the center specimens of every cluster and all new specimens. The center specimen of a cluster is the specimen that has the minimum sum of the square of the distance to other specimens in the cluster. We assign each new specimen to its nearest cluster. To determine where the new specimens should be inserted, we re-create the phylogenetic tree of each cluster using the neighbor-joining method. Note that the distance matrix of the original cluster has been calculated; therefore, the new distance matrix only requires an extremely small amount of NCD computations. The pseudocode of the algorithm is shown in Algorithm 5.

I would like to thank Professor Jun'ichi Takeuchi, who has supported my research with his wealth of knowledge for five years. Professor Takeuchi always helped me with the most difficult parts of my research, I couldn't have been able to finish my doctoral course without his help. I would like to thank Doctor Chansu Han. We worked together for many years and he always gave me precious advice about my research. I would also like to thank my advisers Professor Shuji Kijima and Professor Noboru Murata for their precious suggestions and insightful comments.




(a) clustering accuracy (b) computation rate of NCD matrix

FIGURE 4.1: Results of online processing algorithm compared with batch processing

**Algorithm 5** Algorithm for online processing

**Require:** Phylogenetic tree $T$, clustering result $Tc$, new malware set $L$
**Ensure:** New phylogenetic tree $T'$

```
S = ∅
for each T_i ∈ Tc do
    S = S ∪ {e}, where e is the central malware of cluster T_i
end for
calculate the distances d_ij for (i, j) ∈ S × L
for each malware l ∈ L do
    insert l into nearest cluster
end for
for each T_i ∈ Tc do
    re-construct the phylogenetic tree of T_i
end for
join all clusters into new phylogenetic tree T'
```

### 4.1.1 Experiment of online processing

We evaluated the online processing algorithm with the same data in last section that contained 65,494 IoT malware. For the purpose of simulating the continuous addition of new specimens to the dataset, we used 54% specimens to construct the phylogenetic tree and added 9% specimens into the phylogenetic tree using our proposed algorithm at one time. After online processing for four times, which adds 36% specimens into the phylogenetic tree, we used the last 10% specimens to test the clustering accuracy. Fig. 4.1(a) shows that our online processing algorithm achieved

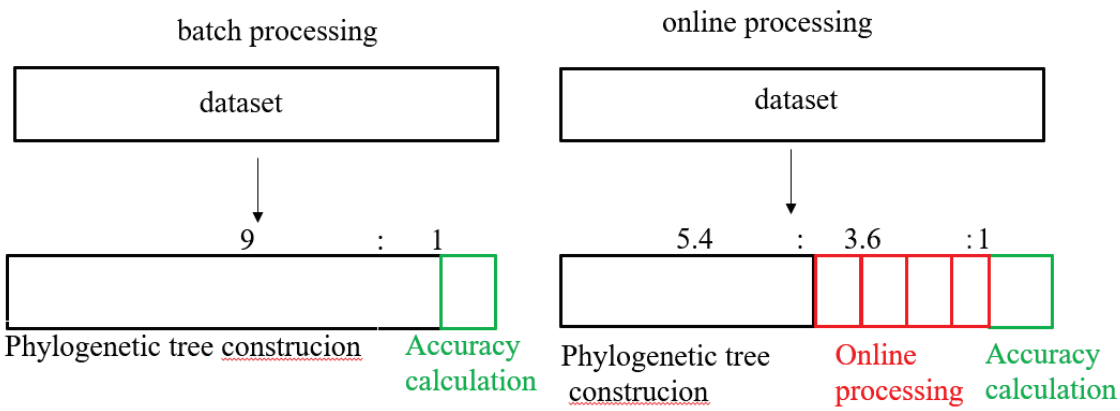



FIGURE 4.2: dataset usage of batch processing and online processing

close accuracy compared to batch processing, the clustering accuracy was only reduced by about 1%. Fig. 4.1(b) shows that our online processing algorithm not only saved the time of re-construct a new phylogenetic tree but also reduced the computational cost of the NCD matrix. When $m_j$ = 7, the malware family name accuracy of batch processing was 94.1%, the accuracy of the online processing was 93.2%, and the computation rate of the NCD matrix was reduced from 2.48% to 1.66%. In Fig. 4.1(b), batch processing is a straight line because its computational cost does not rely on clustering, in another word, the parameter $m_j$ will not affect its computational cost. The computation rate is determined when the phylogenetic tree is constructed.

Our online processing algorithm saves the time of reconstructing the phylogenetic every time some new specimens are added to the dataset while maintaining the clustering accuracy.

We also investigated if the size of the phylogenetic tree or the properties of the specimens to be inserted will influence the clustering accuracy. We designed two different kind of experiments.

In experiment 1, we investigated how the size of the phylogenetic tree affects the online processing algorithm's accuracy. The specimens for constructing the phylogenetic tree and the specimens to be inserted into the phylogenetic using the online processing algorithm are both randomly

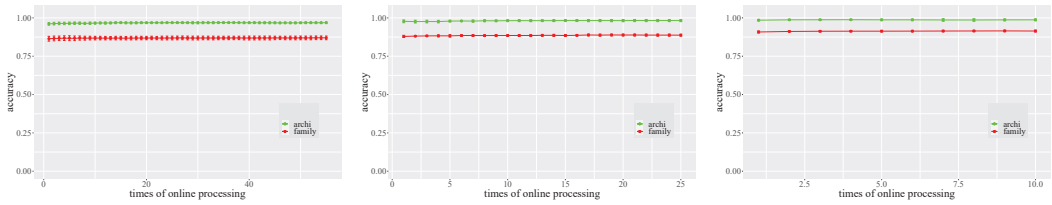
selected from our 65,494 specimens dataset. The size of the phylogenetic tree and the size of the specimens to be inserted to the phylogenetic tree at one time are set up as follows:

- 1) 5000 and 1000, the experiment result is shown in Fig. 4.4(a).
- 2) 10000 and 2000, the experiment result is shown in Fig. 4.4(b).
- 3) 20000 and 4000, the experiment result is shown in Fig. 4.4(c).

The larger size of the phylogenetic tree is, the higher clustering accuracy is achieved. This is because the larger size of the phylogenetic tree contains more clusters, which provides a higher probability for the test malware to find the nearer cluster. But the clustering accuracy did not fall as the online processing going on. We believed this is because that the online processing malware set has the similar properties with the malware in the phylogenetic tree.

| number of specimens | | 2006-2015 | 2016 | 2017_1-4 | 2017_5 | 2017_6 | 2017_7 | 2017_8 | 2017_9 | 2017_10 | 2017_11 | 2017_12 | 2018_1 | 2018_2 | 2018_3 | 2018_4 | 2018_5 | 2018_6 | 2018_7 | 2018_8 | 2018_9 | 2018_10 | 2018_11 | 2018_12 | 2019_1 | 2019_2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1194 | 2006-2015 | 0.9039 | 0.9555 | 0.9346 | 0.9894 | 0.9908 | 0.9776 | 0.9900 | 0.9917 | 0.9935 | 0.9933 | 0.9834 | 0.9905 | 0.9932 | 0.9931 | 0.9905 | 0.9921 | 0.9928 | 0.9924 | 0.9939 | 0.9940 | 0.9942 | 0.9913 | 0.9915 | 0.9885 | 0.9919 |
| 484 | 2016 | 0.9555 | 0.9465 | 0.9595 | 0.9835 | 0.9830 | 0.9784 | 0.9821 | 0.9819 | 0.9833 | 0.9844 | 0.9812 | 0.9825 | 0.9852 | 0.9872 | 0.9868 | 0.9871 | 0.9890 | 0.9888 | 0.9902 | 0.9884 | 0.9881 | 0.9880 | 0.9893 | 0.9882 | 0.9891 |
| 182 | 2017_1-4 | 0.9346 | 0.9595 | 0.9185 | 0.9843 | 0.9865 | 0.9785 | 0.9876 | 0.9890 | 0.9904 | 0.9903 | 0.9816 | 0.9876 | 0.9908 | 0.9899 | 0.9891 | 0.9905 | 0.9916 | 0.9913 | 0.9921 | 0.9913 | 0.9918 | 0.9899 | 0.9905 | 0.9878 | 0.9908 |
| 332 | 2017_5 | 0.9894 | 0.9835 | 0.9843 | 0.9128 | 0.9030 | 0.9324 | 0.9030 | 0.9003 | 0.9012 | 0.9062 | 0.9111 | 0.9030 | 0.9238 | 0.9442 | 0.9526 | 0.9486 | 0.9526 | 0.9475 | 0.9554 | 0.9407 | 0.9343 | 0.9548 | 0.9631 | 0.9603 | 0.9563 |
| 305 | 2017_6 | 0.9908 | 0.9830 | 0.9865 | 0.9030 | 0.8611 | 0.9067 | 0.8650 | 0.8679 | 0.8695 | 0.8759 | 0.8817 | 0.8693 | 0.9002 | 0.9317 | 0.9487 | 0.9406 | 0.9501 | 0.9402 | 0.9482 | 0.9273 | 0.9170 | 0.9471 | 0.9588 | 0.9545 | 0.9487 |
| 186 | 2017_7 | 0.9776 | 0.9784 | 0.9785 | 0.9324 | 0.9067 | 0.9241 | 0.9073 | 0.9079 | 0.9096 | 0.9145 | 0.9166 | 0.9095 | 0.9308 | 0.9518 | 0.9619 | 0.9564 | 0.9626 | 0.9565 | 0.9624 | 0.9491 | 0.9430 | 0.9613 | 0.9690 | 0.9661 | 0.9628 |
| 364 | 2017_8 | 0.9900 | 0.9821 | 0.9876 | 0.9030 | 0.8650 | 0.9073 | 0.8601 | 0.8598 | 0.8639 | 0.8732 | 0.8797 | 0.8667 | 0.8972 | 0.9293 | 0.9447 | 0.9366 | 0.9482 | 0.9371 | 0.9452 | 0.9239 | 0.9145 | 0.9453 | 0.9576 | 0.9532 | 0.9470 |
| 1167 | 2017_9 | 0.9917 | 0.9819 | 0.9890 | 0.9003 | 0.8679 | 0.9079 | 0.8598 | 0.8477 | 0.8551 | 0.8695 | 0.8779 | 0.8645 | 0.8911 | 0.9236 | 0.9355 | 0.9280 | 0.9415 | 0.9310 | 0.9400 | 0.9179 | 0.9101 | 0.9412 | 0.9539 | 0.9495 | 0.9428 |
| 746 | 2017_10 | 0.9935 | 0.9833 | 0.9904 | 0.9012 | 0.8695 | 0.9096 | 0.8639 | 0.8551 | 0.8559 | 0.8690 | 0.8768 | 0.8633 | 0.8934 | 0.9260 | 0.9398 | 0.9321 | 0.9445 | 0.9333 | 0.9423 | 0.9203 | 0.9105 | 0.9419 | 0.9545 | 0.9504 | 0.9436 |
| 849 | 2017_11 | 0.9933 | 0.9844 | 0.9903 | 0.9062 | 0.8759 | 0.9145 | 0.8732 | 0.8695 | 0.8690 | 0.8749 | 0.8831 | 0.8705 | 0.9010 | 0.9321 | 0.9471 | 0.9397 | 0.9494 | 0.9393 | 0.9482 | 0.9280 | 0.9172 | 0.9460 | 0.9576 | 0.9537 | 0.9476 |
| 1760 | 2017_12 | 0.9834 | 0.9812 | 0.9816 | 0.9111 | 0.8817 | 0.9166 | 0.8797 | 0.8779 | 0.8768 | 0.8831 | 0.8860 | 0.8754 | 0.9065 | 0.9354 | 0.9495 | 0.9426 | 0.9526 | 0.9436 | 0.9516 | 0.9326 | 0.9219 | 0.9486 | 0.9593 | 0.9553 | 0.9501 |
| 3308 | 2018_1 | 0.9905 | 0.9825 | 0.9876 | 0.9030 | 0.8693 | 0.9095 | 0.8667 | 0.8645 | 0.8633 | 0.8705 | 0.8754 | 0.8615 | 0.8970 | 0.9292 | 0.9455 | 0.9374 | 0.9495 | 0.9390 | 0.9475 | 0.9264 | 0.9144 | 0.9444 | 0.9563 | 0.9521 | 0.9459 |
| 549 | 2018_2 | 0.9932 | 0.9852 | 0.9908 | 0.9238 | 0.9002 | 0.9308 | 0.8972 | 0.8911 | 0.8934 | 0.9010 | 0.9065 | 0.8970 | 0.9082 | 0.9328 | 0.9334 | 0.9326 | 0.9434 | 0.9442 | 0.9545 | 0.9381 | 0.9307 | 0.9499 | 0.9592 | 0.9563 | 0.9517 |
| 801 | 2018_3 | 0.9931 | 0.9872 | 0.9899 | 0.9442 | 0.9317 | 0.9518 | 0.9293 | 0.9236 | 0.9260 | 0.9321 | 0.9354 | 0.9292 | 0.9328 | 0.9422 | 0.9383 | 0.9412 | 0.9505 | 0.9562 | 0.9647 | 0.9541 | 0.9497 | 0.9597 | 0.9655 | 0.9639 | 0.9610 |
| 301 | 2018_4 | 0.9905 | 0.9868 | 0.9891 | 0.9526 | 0.9487 | 0.9619 | 0.9447 | 0.9355 | 0.9398 | 0.9471 | 0.9495 | 0.9455 | 0.9334 | 0.9383 | 0.9072 | 0.9229 | 0.9347 | 0.9542 | 0.9680 | 0.9596 | 0.9582 | 0.9598 | 0.9641 | 0.9634 | 0.9613 |
| 375 | 2018_5 | 0.9921 | 0.9871 | 0.9905 | 0.9486 | 0.9406 | 0.9564 | 0.9366 | 0.9280 | 0.9321 | 0.9397 | 0.9426 | 0.9374 | 0.9326 | 0.9412 | 0.9229 | 0.9284 | 0.9407 | 0.9540 | 0.9655 | 0.9559 | 0.9536 | 0.9594 | 0.9645 | 0.9634 | 0.9608 |
| 635 | 2018_6 | 0.9928 | 0.9890 | 0.9916 | 0.9526 | 0.9501 | 0.9626 | 0.9482 | 0.9415 | 0.9445 | 0.9494 | 0.9526 | 0.9495 | 0.9434 | 0.9505 | 0.9347 | 0.9407 | 0.9397 | 0.9526 | 0.9645 | 0.9573 | 0.9580 | 0.9631 | 0.9673 | 0.9664 | 0.9648 |
| 699 | 2018_7 | 0.9924 | 0.9888 | 0.9913 | 0.9475 | 0.9402 | 0.9565 | 0.9371 | 0.9310 | 0.9333 | 0.9393 | 0.9436 | 0.9390 | 0.9442 | 0.9562 | 0.9542 | 0.9540 | 0.9526 | 0.9508 | 0.9603 | 0.9514 | 0.9515 | 0.9640 | 0.9698 | 0.9681 | 0.9655 |
| 411 | 2018_8 | 0.9939 | 0.9902 | 0.9921 | 0.9554 | 0.9482 | 0.9624 | 0.9452 | 0.9400 | 0.9423 | 0.9482 | 0.9516 | 0.9475 | 0.9545 | 0.9647 | 0.9680 | 0.9655 | 0.9645 | 0.9603 | 0.9619 | 0.9567 | 0.9582 | 0.9702 | 0.9752 | 0.9738 | 0.9714 |
| 416 | 2018_9 | 0.9940 | 0.9884 | 0.9913 | 0.9407 | 0.9273 | 0.9491 | 0.9239 | 0.9179 | 0.9203 | 0.9280 | 0.9326 | 0.9264 | 0.9381 | 0.9541 | 0.9596 | 0.9559 | 0.9573 | 0.9514 | 0.9567 | 0.9410 | 0.9433 | 0.9611 | 0.9684 | 0.9662 | 0.9625 |
| 1799 | 2018_10 | 0.9942 | 0.9881 | 0.9918 | 0.9343 | 0.9170 | 0.9430 | 0.9145 | 0.9101 | 0.9105 | 0.9172 | 0.9219 | 0.9144 | 0.9307 | 0.9497 | 0.9582 | 0.9536 | 0.9580 | 0.9515 | 0.9582 | 0.9433 | 0.9328 | 0.9557 | 0.9634 | 0.9603 | 0.9562 |
| 7847 | 2018_11 | 0.9913 | 0.9880 | 0.9899 | 0.9548 | 0.9471 | 0.9613 | 0.9453 | 0.9412 | 0.9419 | 0.9460 | 0.9486 | 0.9444 | 0.9499 | 0.9597 | 0.9598 | 0.9594 | 0.9631 | 0.9640 | 0.9702 | 0.9611 | 0.9557 | 0.9627 | 0.9674 | 0.9664 | 0.9634 |
| 17948 | 2018_12 | 0.9915 | 0.9893 | 0.9905 | 0.9631 | 0.9588 | 0.9690 | 0.9576 | 0.9539 | 0.9545 | 0.9576 | 0.9593 | 0.9563 | 0.9592 | 0.9655 | 0.9641 | 0.9645 | 0.9673 | 0.9698 | 0.9752 | 0.9684 | 0.9634 | 0.9674 | 0.9698 | 0.9694 | 0.9670 |
| 16289 | 2019_1 | 0.9885 | 0.9882 | 0.9878 | 0.9603 | 0.9545 | 0.9661 | 0.9532 | 0.9495 | 0.9504 | 0.9537 | 0.9553 | 0.9521 | 0.9563 | 0.9639 | 0.9634 | 0.9634 | 0.9664 | 0.9681 | 0.9738 | 0.9662 | 0.9603 | 0.9664 | 0.9694 | 0.9678 | 0.9655 |
| 12463 | 2019_2 | 0.9919 | 0.9891 | 0.9908 | 0.9563 | 0.9487 | 0.9628 | 0.9470 | 0.9428 | 0.9436 | 0.9476 | 0.9501 | 0.9459 | 0.9517 | 0.9610 | 0.9613 | 0.9608 | 0.9648 | 0.9655 | 0.9714 | 0.9625 | 0.9562 | 0.9634 | 0.9670 | 0.9655 | 0.9615 |

FIGURE 4.3: Average distance among specimens in different period




(a) clustering accuracy (b) computation rate of NCD matrix (c) computation rate of NCD matrix

FIGURE 4.4: Results of online processing experiment 1

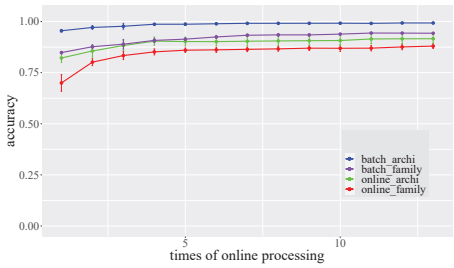



FIGURE 4.5: Result of online processing experiment 2

In the experiment 2, we investigated how the properties of the new specimens affects the online processing algorithm's accuracy. Fig. 4.3 shows the average distance among specimens in different periods. We can see from Fig. 4.3 that specimens collected from May 2017 to February 2018 have smaller distances. And the distances between specimens collected from March 2018 to February 2019 and specimens collected from May 2017 to February 2018 are larger, which means

specimens in these two periods have different properties. Therefore, in the second experiment, we choose the specimens collected from May 2017 to February 2018 for constructing the phylogenetic tree and the specimens collected from March 2018 to February 2019 for online processing. The result is shown in Fig. 4.5: if the properties of the online processing specimens changed from the original specimens, the clustering accuracy of the family name was decreased to under 70%. The accuracies rise as the online processing goes on because the more specimens are inserted into the phylogenetic tree, the more similar the specimens in the phylogenetic tree and the specimens to be inserted are. After inserting enough specimens, the specimens in the phylogenetic tree and the specimens to be inserted will have similar properties, which lead to the accuracies being steady, like in experiment 1. The purple and blue lines show the clustering accuracies when recreating the phylogenetic tree after adding new specimens to the dataset (batch processing). Therefore, when the properties of the online processing specimens change from the original specimens, a recreation of the phylogenetic tree to maintain the clustering accuracy is needed.

## 4.2 Attempts on New Seed Set Selection Method

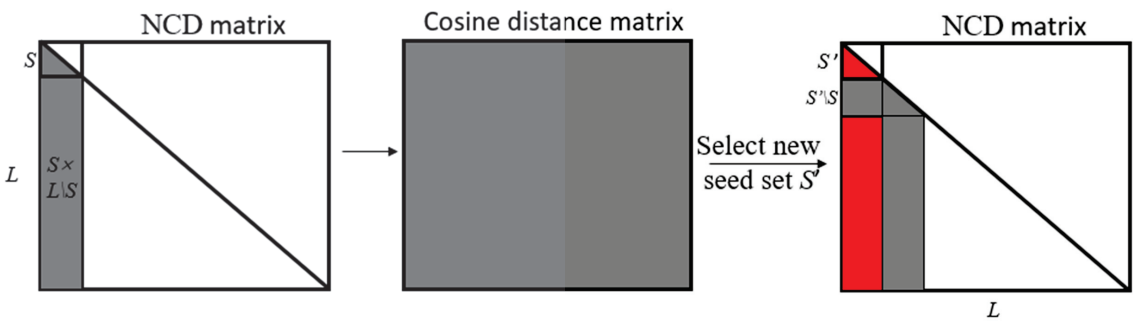



FIGURE 4.6: Procedure of the new seed set selection

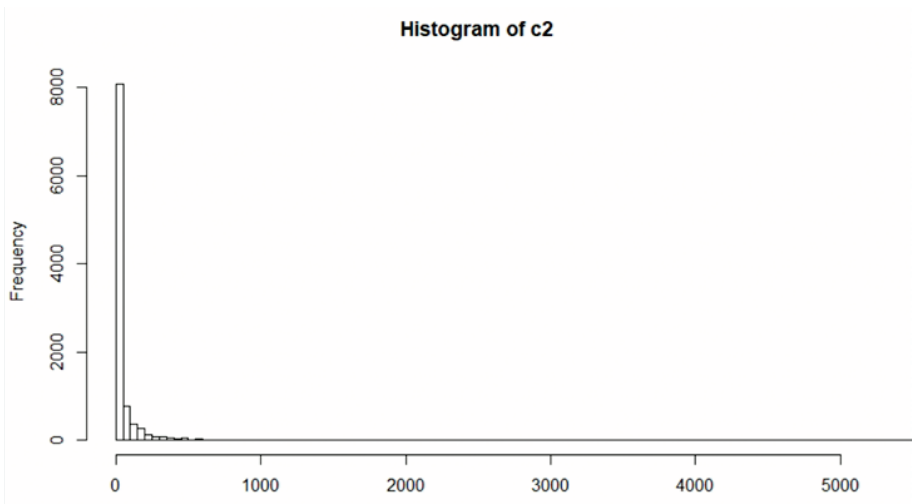



FIGURE 4.7: Result of comparison of NCD and cosine distance

Our method introduced in section 2 supports large-scale dataset by randomly calculating only a small part of the distance matrix. The idea of our method is similar to prototype-based clustering algorithm. We first randomly select a seed set with size k and create a phylogenetic tree with it. The remaining malware specimens are associated with their closest specimen (prototype) in the seed set. Then create a phylogenetic tree for each prototype group and finally join them together. We are now trying to design a new, smarter way to choose where of the distance matrix to calculate.

The selection of the seed set is important because it is the core of our phylogenetic tree. The seed set should represent the whole malware set. Therefore, we tried a new method of seed set selection. We assume that we do not have any prior knowledge of the malware set, so we have to use the calculated distances for seed set selection. The diagram of the new seed set selection in shown in Fig 4.6. After associating the remaining specimens to their closest prototype, we have calculated a distance matrix of size $N \times k$. Regard each row of the $N \times k$ matrix as a vector, we can get the cosine distance matrix of these vectors. The purpose of this step is to get a complete distance matrix.

To investigate if the cosine distance matrix can represent the NCD matrix, we used the calculated $71410 \times 71410$ NCD matrix in a small test experiment. We randomly choose a malware specimen $A$, and find its closest specimen $B$ in the cosine matrix. Then we find out what number is $B$ in the closest sort of $A$ in the NCD matrix. We tried this 10,000 times and the results and showed in Fig 4.7. For any specimen, its closest specimen in the cosine matrix has a 91.58% possibility to be the 1% closest specimen in the NCD matrix. And has a 99.39% possibility to be the 5% closest specimen in the NCD matrix. Therefore, we can say that the cosine matrix can represent the NCD matrix to some extent.

Then, we apply the simple greedy algorithm to choose the $k$ specimens that are far from each other as the new seed set. The pseudocode of the simple greedy algorithm is shown in Algorithm 6.

### 4.2.1 Experiment of the new seed selection method

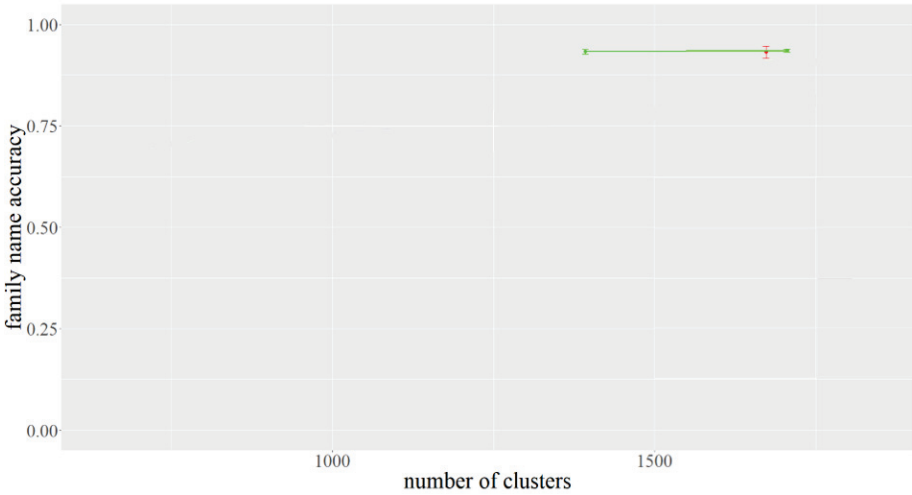



FIGURE 4.8: Result of the new seed selection method distance

We used the same dataset in Table 3.1 that contained 65,494 IoT malware and applied the 10-fold cross-validation to evaluate the clustering accuracy. We compared the new seed set selection method (represented as random+cos) with the original one (represented as random). The results are shown in Fig 4.6. Two methods achieved the same level of clustering accuracy and the new method achieved a much smaller variance. We considered the reason for clustering accuracy not improving maybe that the simple greedy algorithm tends to choose specimens from the corner of the dataset, they may be far from each other but can not well represent the whole dataset. In our future work, we are going to attempt other seed selection methods to achieve better clustering accuracy.

**Algorithm 6** Simple greedy algorithm

**Require:** a finite dataset $V$
**Ensure:** a seed set $C$ with size $k$
1: initialization of $C$: randomly choose one data in $V$
2: **while** $|C| < k$ **do**
3: $v = \max_{v \in V} d(v, C)$, where $d(v, C) = \min_{c \in C} dist(v, c)$, where $dist(v, c)$ is the distance between $v$ and $c$.
4: add $v$ to $C$
5: **end while**
6: return $C$

## 4.3 Investigation about user-code-only binary

There are two ways for libraries to be linked with the malware specimen, statically-linked, and dynamic-linked. The statically-linked library will be built inside the malware's binary, which will influence the NCD's calculation. Therefore, in this topic, we will investigate how the statically-linked library influences the NCD between malware specimens by removing the statically-linked library using the *IDA Pro*[1].

TABLE 4.1: Breakdown of ISA and malware family names of the dataset

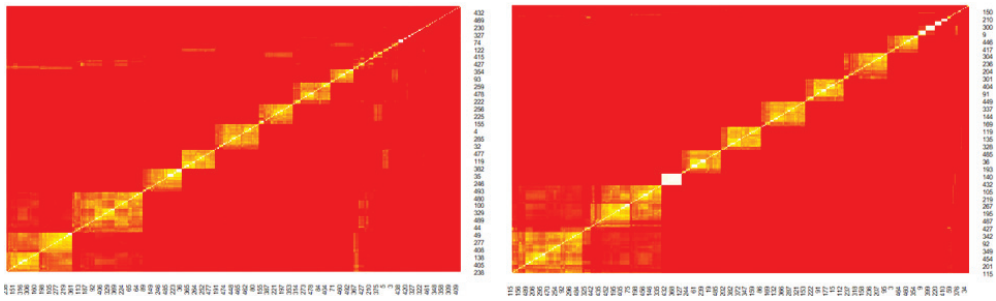
| ISA | Malware Family | | | | Total |
|---|---|---|---|---|---|
| | Bashlite | Mirai | Tsunami | others | |
| ARM | 288 | 6345 | 6 | 3898 | 10537 |
| MIPS | 181 | 3578 | 3 | 1730 | 5492 |
| Intel | 58 | 1932 | | 617 | 2607 |
| x86 | 11 | 112 | 3 | 34 | 160 |
| PowerPC | 35 | 1725 | 1 | 883 | 2644 |
| Renesas | 50 | 1738 | 3 | 868 | 2908 |
| SPARC | | 1 | | | 1 |
| Motorola | 52 | 1701 | 2 | 858 | 2867 |
| Total | 675 | 17132 | 18 | 8888 | 26713 |

We used the IoT malware collected from IoTpot [27] and did some small experiments. The collection period was from May 2020 to July 2020. The breakdown of ISA and malware family names of the dataset in shown in Table 4.1 First, we randomly chose 500 malware specimens. We calculated the NCD matrix of the specimens and the NCD matrix of the user-code-only specimens. The heatmaps of the two NCD matrix is shown in Fig 4.9. Note that these two graphs have been resorted to form clusters. We can see from Fig 4.9 that many original binaries had no neighbors, they were far from all other specimens' binaries and did not belong to any cluster. After removing the libraries from the binaries, the number of no-neighbor specimens was significantly reduced. Moreover, some white clusters were formed, which means after removing the libraries, these specimens had totally the same binaries.

Secondly, we chose 281 Mirai-architecture malware and calculated their NCD matrix. The heatmaps of the two NCD matrix is shown in Fig 4.10. In general, the distances in the NCD matrix became larger after removing the libraries. This is because these specimens were originally close to each other because of the same libraries. The user code parts of the binaries are actually more different.

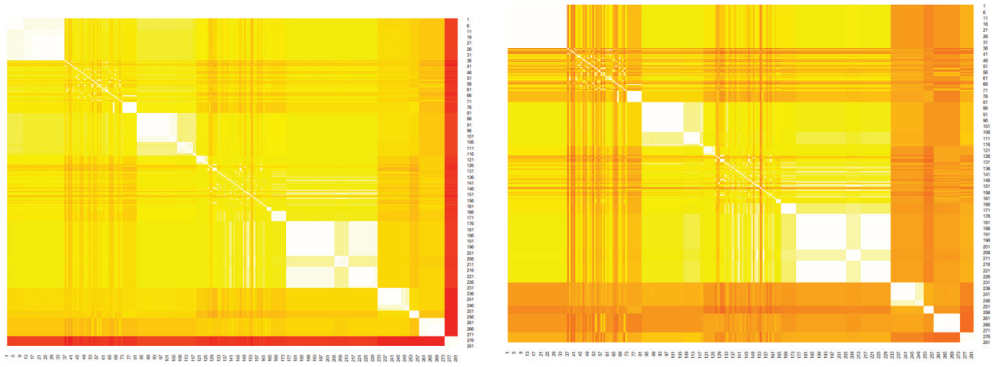
To summarize this investigation, the user-code-only binary can help us find some hidden similarities and better distinguish the close specimens pairs with the far pairs. In general, the user-code-only NCD matrix can better represent the similarity of the malware specimens.

[1] https://hex-rays.com/ida-pro/



(a) original specimens (b) user-code-only specimens

FIGURE 4.9: Heatmaps of the NCD matrix of randomly selected 500 specimens



(a) original specimens (b) user-code-only specimens

FIGURE 4.10: Heatmaps of the NCD matrix of 281 Mirai specimens

# Chapter 5

# Hierarchical Clustering of IoT Malware Using Active Data Selection

In the second section, we introduced our scalable algorithm (hereinafter referred to as FACCP) that construct a phylogenetic tree by randomly calculating only a small part of the distance matrix. In this section, rather than randomly calculating the distance matrix, we choose to smartly decide where to calculate by applying the active learning [15].

Our scalable method introduced in the second section supports large-scale data by randomly calculating only a small part of the distance matrix. Construction of a phylogenetic tree based on the FACCP scalable method can be regarded as regression model learning, that is, supervised learning, and distance data can be regarded as its attribute (covariate). Therefore, because this is similar to active learning, we would like to adopt active learning to the clustering to improve the distance calculation reduction rate. In fact, a clustering method by active learning has been proposed [16, 45]. And it is applicable if we skip the construction of the phylogenetic tree, which is an intermediate stage, making the clustering process simpler in the meantime. In this paper, we applied active learning (also called query learning) to reduce the similarity calculation cost (compression attempts). We believe active learning can achieve even better reduction by smartly choosing which part of the distance matrix to calculate. The key idea of active learning is that the learning algorithm can smartly select the data has most information to observe. Active learning works well for these cases that labeled data is difficult, time-consuming, or expensive to obtain because active learning needs much lesser labeled data than a normal machine learning algorithm [37].

We applied the hierarchical clustering method in [45] which is to clustering all the specimens into a binary tree. All the specimens are assigned to a leaf cluster. The optimizer algorithm Mean-Field Annealing (MFA) determines the assignments by minimizing the cost function. The cost function is defined to measure the compactness of the clusters. Finally, the active data selection criterion decides the next observation position based on the current clustering and observation information. MFA and active data selection are repeated until the cost function converges.

We evaluated the active data clustering with a Linux malware specimen set (mostly IoT malware) containing 3,008 specimens collected from VirusTotal. We used this dataset to evaluate the reduction of the active clustering algorithm's calculation cost and clustering accuracy. For performance comparison, we choose MFA combines random sampling as the benchmark. The results showed that active learning has better performance than the random sampling. We confirmed that where to observe in the distance matrix is an important factor of algorithm's performance.

## 5.1 Active Clustering Algorithm

In this section, we introduce the active clustering algorithm from [45]. This algorithm consists of two parts: clustering and active data selection. The clustering algorithm determines the assignment matrix based on the current observation of the distance matrix. The active data selection

determines the next observation based on the current assignment matrix. The algorithm starts with 1% of the distance matrix is observed randomly.

**Problem Setting**

The distance matrix of the dataset is represented as $\mathbf{D} \in \mathbb{R}^{N\times N}$, as mentioned in the introduction, only small parts of $\mathbf{D}$ are calculated, which leaves most parts of $\mathbf{D}$ to remain unknown. $\mathcal{N}_1$, ..., $\mathcal{N}_N$ record which distances have been calculated: if $D_{ij}$ has been calculated, then $j \in \mathcal{N}_i$.

- Assignment Matrix

The hierarchical clustering model in [45] is based on a binary tree, which means all the specimens are assigned to a leaf cluster. Clustering $N$ data into leaf clusters can be seen as an assignment problem. Let $K$ be the depth of the binary tree $T$ (the depth of root node is 0), so the number of leaf nodes is $2^K$, and the total number of nodes is $2^{K+1} - 1$. Clustering $N$ data equals assign $N$ data into $2^K$ leaf nodes. Clustering can be encoded as an assignment matrix $\mathbf{M} \in \{0,1\}^{N\times(2^{K+1}-1)}$, where $M_{ij} = 1$ means *i-th* data is in *j-th* cluster. The inner nodes are also treated as clusters. An inner node cluster consists of the data that belong to its successor leaf nodes. $2^k th$ column to $(2^{k+1} - 1)th$ column of $\mathbf{M}$ represent the *k-th* layer's assignment of the binary tree. $\mathbf{M}$ has following properties:
(a) any data should be assigned to one cluster

$$\forall i, \sum_{j=2^K}^{2^{K+1}-1} M_{ij} = 1.$$

(b) inner nodes' assignments are inherited upwards in the tree

$$\forall i, \sum_{j=1}^{2^K-1} M_{ij} = M_{i(2j)} + M_{i(2j+1)}.$$

(c) root node contains all the data

$$\sum_{i=1}^{N} M_{i1} = N.$$

(d) each cluster contains at least one data

$$\forall j, \sum_{i=1}^{N} M_{ij} \geq 1.$$

- Cost Function

To measure how well an assignment matrix is, we apply the cost function in [29,45] :

$$H(\mathbf{M}, \mathbf{D}, \mathcal{N}, K) = \sum_{k=1}^{K} W(k) \sum_{i=1}^{N} \sum_{v=2^k}^{2^{k+1}-1} M_{iv} d_{iv}, \tag{5.1}$$

$$\text{where } d_{iv} = \frac{\sum_{j\in\mathcal{N}_i} M_{jv} D_{ij}}{\sum_{j\in\mathcal{N}_i} M_{jv}}. \tag{5.2}$$

Here $d_{iv}$ represents the distance between data $i$ and cluster $v$. $H$ sums up the sum of the distances in the clusters with
weight $W$, respectively. $H$ measures the compactness of the clusters, the smaller the cost function, the better the clustering is.

**Algorithm 7** Mean-Field Annealing (MFA)

**Input:** Temperature schedule vector: T
**Output:** $\mathbf{M} \in \{0,1\}^{\mathrm{N}\times 2^{\mathrm{K}+1}-1}$

```
Initialization: randomly generate a assignment matrix M
 for t from 1 to length(T) do
    while cost function is decreasing do
        randomly select i-th data
          calculate the mean field vector of i: φ_ik = −∂H/∂M_ik (k from 2^K to 2^(K+1) − 1)
          update the assignment matrix: M_ik = e^(φ_ik/T_t) / Σ_{n=2^K}^{2^(K+1)−1} e^(φ_in/T_t)
    end
    t=t+1
end
return(M)
```

## 5.2 Clustering

The clustering algorithm finds an assignment matrix **M** that minimizes the cost function $H$, which can be considered as a combinatorial optimization problem. MFA is applied to solve the optimization problem.

MFA combines mean-field theory and simulated annealing. Mean-field theory simplifies a high-dimensional complex random model to a simpler model. The complex model usually contains many molecules that interact with each other. Instead of calculating the enormous interactions, one molecule's interactions with other molecules are approximated as an averaged interaction. As a result, the study of complex systems or models is possible.

Simulated annealing is a well-known method for solving optimization problems. Annealing is a metallurgy process that metal cools down slowly so that molecules can find places that minimize the internal energy, which leads to steady. The principle of simulated annealing is similar to annealing: The solution is consists of molecules' states, and it's randomly moving in the solution space. A better solution (smaller value of energy function) is always accepted. The key idea is that a worse solution is accepted under certain probability. This probability becomes smaller as the temperature gets lower, which allows the algorithm to jump out of the local minimum solution and finally converge to the global minimum solution.

The pseudocode of MFA is shown in Algorithm 7. Note that $M_{ij}$ can take any real value between 0 and 1 during the cooling. $M_{ij}$ represents the probability of assigning $i$ to cluster $j$. When the temperature cools down and the system becomes stabilized, $M_{ij}$ will converges to 0 or 1.

## 5.3 Active Data Selection

After MFA determines the best assignment matrix based on the current observation of the distance matrix, we need to decide where to observe next. The active data selection algorithm finds where it expects to have the most information. To measure the expected value of information, we applied the Bayesian approach developed in [23], which is also the method applied in [16, 45]. If all the distances have been observed, the distance between data $i$ and cluster $\nu$ is given by

$$d^*_{i\nu} = \frac{\sum_{j=1}^{N} M_{j\nu} D_{ij}}{\sum_{j=1}^{N} D_{ij}}.$$

But since we are using incomplete data, $d^*_{i\nu}$ can not be calculated in realistic. Therefore, we consider $d^*_{i\nu}$ as a random variable and rely on its point estimator $d_{i\nu}$ in (5.2). Assume the prior distribution of $d^*_{i\nu}$ a noninformative prior distribution. $d_{i\nu}$ is the sample mean, $\sigma^2_{i\nu}$ is the sample variance and $m_{i\nu}$ is the number of sample. The marginal posterior distribution of $d^*_{i\nu}$ is a student t distribution:

$$t = \sqrt{m_{i\nu}}(d^*_{i\nu} - d_{i\nu})/\sigma_{i\nu}.$$

Thus $d^*_{i\nu}$ can be expressed as:

$$d^*_{i\nu} = d_{i\nu} - t * \frac{\sigma^2_{i\nu}}{m_{i\nu}}.$$

The probability density function of $d^*_{i\nu}$ is denoted by $f_{i\nu}(d^*_{i\nu}|d_{i\nu}, \sigma_{i\nu}, m_{i\nu})$. Furthermore, the *Expected Value of Perfect Information* (EVPI) [16] can be defined with $f_{i\nu}$:

$$\begin{aligned} \text{EVPI} = &\int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} \max_{\nu}\{d^*_{i\alpha} - d^*_{i\nu}\} \\ &\times \prod_{\nu=2^K}^{2^{K+1}-1} f_{i\nu}(d^*_{i\nu}|d_{i\nu}, \sigma^2_{i\nu}, m_{i\nu}) d\, d^*_{i\nu}. \end{aligned} \tag{5.3}$$

where $\alpha = \arg\min_{\nu} d_{i\nu}$. The EVPI measures the loss expects to incur by making the decision $\alpha$ based on the incomplete information $d_{i\nu}$ instead of the optimal decision $\alpha^*$. In other words, the expected gain if $\alpha^*$ was revealed to us.

However, in the actual experiment, the perfect information can not be obtained. Thus, what we are more interested in is the *Expected Value of Sampling Information* (EVSI). The EVSI measures how much uncertainty is expected to be remained after observing the additional data. In other words, EVPI minus EVSI measures how much gain we are expecting from additional data. The selection strategy is that drawing $m^+_{i\nu}$ additional samples from the *ν-th* cluster, which is expected to gain the most information. Drawing $m^+_{i\nu}$ additional samples will not affect the unbiased estimates $d_{i\nu}$ and $\sigma^2_{i\nu}$, but increase the sample size $m_{i\nu}$. Therefore, the EVSI of drawing samples from cluster $j$ is given by the following equation:

$$\begin{aligned} \text{EVSI}_{ij} = &\int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} \max_{\nu}\{d^*_{i\alpha} - d^*_{i\nu}\} \\ &\times f_{ij}(d^*_{ij}|d_{ij}, \sigma^2_{ij}, m_{ij} + m^+_{ij}) \\ \times &\prod_{\substack{\nu=2^K \\ \nu\neq j}}^{2^{K+1}-1} f_{i\nu}(d^*_{i\nu}|d_{i\nu}, \sigma^2_{i\nu}, m_{i\nu}) d\, d^*_{i\nu}\, d\, d^*_{ij}. \end{aligned} \tag{5.4}$$

To calculate equation (5.4), we apply Monte-Carlo techniques to sampling from the student t distribution to estimate the above integral. Each leaf cluster will calculate this equation, and additional samples will be drawn from the cluster with minimal EVSI value.

The pseudocode of the active clustering algorithm is shown in Algorithm 8.

**Algorithm 8** Active Clustering

**Input:** $N \times N$ unknown distance matrix $\mathbf{D}$,$K$
**Output:** $\mathbf{M} \in \{0,1\}^{N \times 2^{K+1}-1}$
$\mathcal{N} = (\mathcal{N}_1, ..., \mathcal{N}_N)$. If $j \in \mathcal{N}_i$, means the value of $D_{ij}$ is available.
Initialization:
randomly observe 1% of $\mathbf{D}$ ($\mathcal{N}$ will be changed)
**while** *cost function getting smaller* **do**
  using MFA to calculate the $\mathbf{M}$ that minimize cost function $H(\mathbf{M}, \mathbf{D}, \mathcal{N}, K)$
    Active Data Selection :
    **for** *i from 1 to N* **do**
      $v = \arg\min_v \mathrm{EVSI}_{iv}$
      observe $m_{iv}^{+}$ distances between data $i$ and data that belongs to cluster $v$ ($\mathcal{N}$ will be changed)
  **end**
**end**

## 5.4 A Mistake of the Original Paper

In the original paper [45] that proposed this algorithm, the authors said that the sampling decisions would be met for the objects and clusters with the maximal EVSI value. But in fact, the EVSI is not directly calculating the expected value of sampling information, but the uncertainty that is remained after observing the additional data. The sampling decision should be met for objects and clusters with the least uncertainty, which means the minimal EVSI value. We did some small experiments to prove this.

TABLE 5.1: simulated clusters and their EVSI value (case 1)

| cluster | sample mean | sample variance | number of samples | EVSI |
|---|---|---|---|---|
| 1 | 0.7 | 0.09 | 6 | 0.03981 |
| 2 | 0.35 | 0.09 | 6 | 0.03820 |

TABLE 5.2: simulated clusters and their EVSI value (case 2)

| cluster | sample mean | sample variance | number of samples | EVSI |
|---|---|---|---|---|
| 1 | 1 | 0.002 | 6 | 0.22823 |
| 2 | 1 | 0.17 | 6 | 0.22476 |

We can see from Table 5.1, 5.2 and 5.3 that clusters have a larger sample mean, a smaller sample variance and a larger number of samples tend to have a larger EVSI value. Sampling from those clusters will not give us much information or reduce much cluster's uncertainty. Because we can already predict pretty well what we will get from the sampling. Moreover, sampling from the cluster has a larger variance and the number of samples will make its EVSI even larger, which means all the following sampling will be made at this cluster. This is obviously wrong, the sampling decision should be made at the cluster that has minimal EVSI value.

TABLE 5.3: simulated clusters and their EVSI value (case 3)

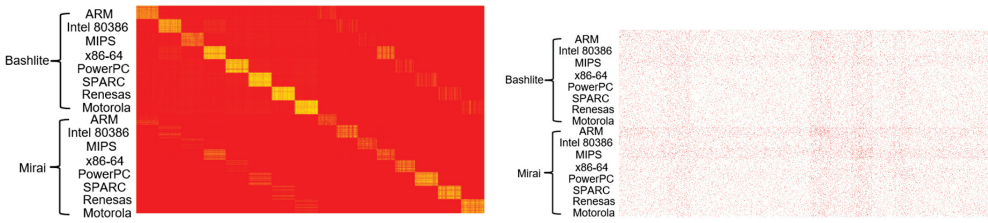
| cluster | sample mean | sample variance | number of samples | EVSI |
|---|---|---|---|---|
| 1 | 0.5 | 0.5 | 4 | 0.53152 |
| 2 | 0.5 | 0.5 | 100 | 0.55409 |

## 5.5 Performance Evaluation with Experiment

In this subsection, we introduce our experiment to evaluate the active clustering algorithm with IoT malware. As a benchmark, we also applied random sampling as the data selection strategy. For both data selection strategies, we applied the MFA as the optimization algorithm. Furthermore, we compared the percentage of the calculation of the distance matrix and clustering accuracy with our former work, a fast algorithm for constructing a phylogenetic tree of malware specimens and clustering [13].

### Dataset

We collected 3,008 Linux malware specimens (mostly IoT malware) from VirusTotal. The collection period was from November 2018 to February 2019. Our dataset is consists of Bashlite and Mirai. The breakdown of the dataset is shown in Table 5.4. The malware family name was decided by AVClass [36]. AVClass is a Python tool to label malware samples using VirusTotal JSON reports.




(a) Distance matrix (b) Sampling snapshot

FIGURE 5.1: Heatmap of the distance matrix and observation position of the distance matrix

### 5.5.1 Experimental Setup

We implemented our algorithm using the programming language `julia`. `xz` command (version 5.1.0 alpha) of Linux was used to compress malware binaries for computing NCDs. This `xz` adopted the Lempel-Ziv-Markov chain algorithm (LZMA) [35] as the compression algorithm. The CPU of our computation environment is 2.6GHz Intel-Xenon-Gold-6126. In the active data selection algorithm, the computation of the Monte-Karlo method was parallel computed with 80 threads.

TABLE 5.4: Breakdown of ISA and malware family names of the dataset

| Malware Family | ISA | | | | | | | | Total |
|---|---|---|---|---|---|---|---|---|---|
| | ARM | MIPS | Intel | x86 | PowerPC | Renesas | SPARC | Motorola | |
| Bashlite | 195 | 194 | 197 | 189 | 197 | 200 | 200 | 200 | 1572 |
| Mirai | 161 | 165 | 183 | 157 | 171 | 199 | 200 | 200 | 1436 |
| Total | 356 | 359 | 380 | 346 | 368 | 399 | 400 | 400 | 3008 |

### 5.5.2 Evaluation Metrics

We evaluate the active clustering algorithm with three metrics.

1. ***Cost function***: We compared how the cost function's value changed as the observation of the distance matrix gradually increase.

2. ***Family name clustering accuracy***: The clustering accuracy is evaluated by 10-fold cross-validation. Specifically, the malware set is divided into ten parts, and clustering is performed using 90% of the specimens. Then assign each remaining 10% of the specimens to their nearest cluster. The cluster's family name is decided by the majority of the specimen's family name in that cluster. The family name clustering accuracy is calculated as the percentage of correct clustered specimens.

3. ***Architecture name clustering accuracy***: The architecture name clustering accuracy is calculated the same as the family name clustering accuracy.

### 5.5.3 Evaluation Results

Fig. 5.1(a) shows the heatmap of the distance matrix, and the red color represents the greater distance (low similarity). Fig. 5.1(b) is the heatmap of the assignment matrix **M**, which shows the observed position after having observed 3% of the distance matrix. Fig. 5.1(a) and Fig. 5.1(b) are both symmetric matrices, and their data are sorted in the same order. The active data selection algorithm chooses the observation mainly from the clusters with low similarity among the cluster. This is because the clusters with high similarity will form clusters successfully at an early phase. There is no need to draw any more samples from those clusters.

Cost functions' change of active data selection and random sampling are shown in Fig. 5.2. Both methods are evaluated on the complete data (100% observation). The cost function of active data selection converges about 20% faster than random sampling, which means the active data selection algorithm requires 20% lesser observation of the distance matrix.

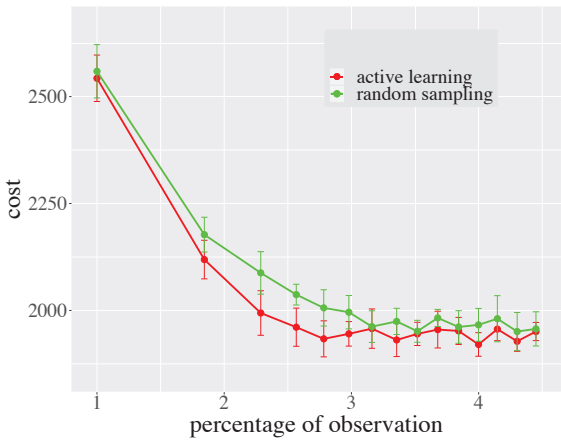
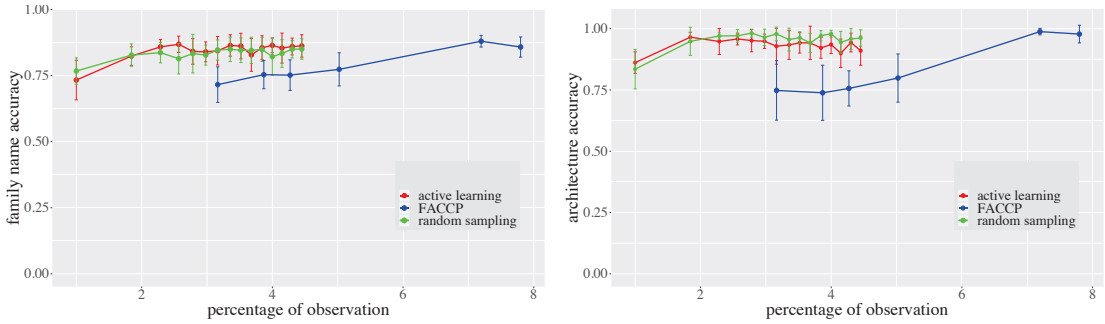



FIGURE 5.2: Cost functions of active data selection and random sampling

Result of the 10-fold cross-validation is shown in Fig. 5.3(a) and Fig. 5.3(b). The active clustering algorithm and random sampling combined with MFA showed the same level of clustering accuracy. Compared to the FACCP, the active clustering algorithm achieved much higher accuracy. In other words, the active clustering algorithm can achieve the same level of accuracy as the FACCP with less than half of the FACCP's observation.

With the experiment results, we confirmed that the active clustering algorithm observed about 20% lesser NCD than the benchmark and achieved the same level of clustering accuracy. Compared to the FACCP, the observation of the NCD matrix decreases from 7.2% to 2.6%. Besides, the




(a) Family name accuracy (b) Architecture name accuracy

FIGURE 5.3: Clustering accuracy

active clustering algorithm achieved much better accuracy than the FACCP at the same observation percentage.

Even though the active data selection algorithm reduces 64% of the calculation amount of NCD than FACCP, the runtime of MFA is very long, which results in that the active clustering algorithm is much slower than the FACCP. Currently, the runtime of the active clustering algorithm is about 7 hours, while the runtime of the FACCP is about 30 minutes. The active data selection algorithm showed excellent performance, but the optimizer algorithm (MFA) is now the whole algorithm's bottleneck.

### 5.5.4 To Overcome the Bottleneck

To solve this problem, we did some research but have not made any breakthroughs. Our approach is to propose a faster, parallelable MFA algorithm [17, 25, 34]. The advantage of this approach is that the runtime can be reduced significantly based on the computer's hardware specifications. In paper [25], Okuyama et.al proposed a parallel optimization algorithm named momentum annealing (MA). They converted an Ising model into a bipartite graph and achieved proofing that the spin configuration on the left (or right) side of Figure 5.4(b) [25] also minimizes the Ising energy of Figure 5.4(a). In Figure 5.4(a), $\sigma 1$ and $\sigma 2$ can not be updated at the same time because their spin states will influence each other. But in Figure 5.4(b), updating all spins on each side in parallel is possible. Okuyama et.al's model is designed for binary optimization, if we want to apply their method to our problem, we need to extend their proof to multiple value cases.

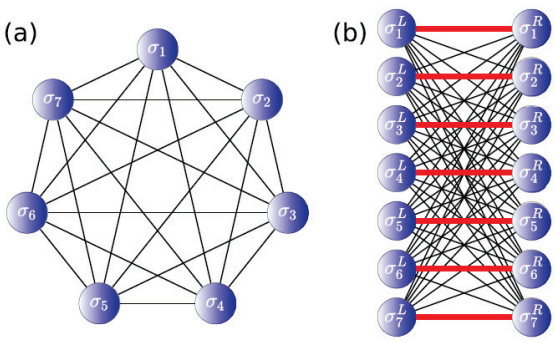



FIGURE 5.4: Conversion of Ising model into bipartite graph

# Chapter 6

# Conclusion

We proposed a fast algorithm for constructing the phylogenetic tree, which significantly reduced the computational cost than the conventional method. We proposed an algorithm applying the `MDL Criterion` to clustering the constructed phylogenetic tree. Furthermore, we proposed an online processing algorithm that can reduce the computation cost in actual operation by skipping the phylogenetic tree reconstruction while maintaining the clustering accuracy. We evaluate our algorithms' scalability and clustering accuracy using a large-scale IoT malware set.

Our experiments using 65,494 IoT malware specimens show that our fast algorithm reduced the computational cost by 97.52 % compared to the neighbor-joining method. By improving the clustering algorithm using `MDL Criterion`, our clustering algorithm achieved significantly higher accuracy than the `Inconsistency Coefficient`. In the best case, the family name clustering accuracy was 95.5% and the architecture name accuracy was 99.3%. Furthermore, the online processing algorithm reduced 33% of the computational cost than the batch processing algorithm while maintaining a clustering accuracy of 94%. Our experiment results show that our method successfully reduces a significant amount of computational cost.

We also evaluated the feasibility and efficiency of the existing active clustering algorithm with our experiment using 3,008 actual Linux malware specimens. And we introduced our attempts at overcoming the bottleneck of the algorithm.

# Acknowledgement

The research for this thesis was conducted under a contract of “MITIGATE” among “Research and Development for Expansion of Radio Wave Resources (JPJ000254)”, which was partly supported by the Ministry of Internal Affairs and Communications, Japan. This work was also partly supported in part by JST SPRING, Grant Number JPMJSP2136.

I want to thank Professor Jun’ichi Takeuchi, who supported my research with his wealth of knowledge for 5 years. He always helped me with the most difficult parts of my research. I couldn’t have finished my doctoral course without his help.

I would like to thank Doctor Chansu Han. We worked together for many years, he gave me lots of precious suggestions, which greatly helped me to finish my work.

I would like to thank the rest of the thesis committee: Prof. Eiji Takimoto and Associate Prof. Daisuke Ikeda for giving me precious suggestions and insightful comments to improve the thesis.

I would also like to express my gratitude to Professor Shuji Kijima and Professor Noboru Murata. As my advisor, They gave me precious suggestions and insightful comments about my research.

At last, I want to thank my families for supporting me all the time.

# Bibliography


[1] Adel Abusitta, Miles Q. Li, and Benjamin C.M. Fung. Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications*, 59:102828, 2021.

[2] Manos Antonakakis, Tim April, Michael Bailey, et al. Understanding the mirai botnet. In *Proceedings of the 26th USENIX Conference on Security Symposium*, pages 1093–1110, 2017.

[3] Michael Bailey, Jon Oberheide, Jon Andersen, Zhuoqing Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. In *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007, Proceedings*, pages 178–197, 2007.

[4] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Krügel, and Engin Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA*, 2009.

[5] Niket Bhodia, Pratikkumar Prajapati, Fabio Di Troia, and Mark Stamp. Transfer learning for image-based malware classification, 2019.

[6] Manuel Cebrian, Manuel Alfonseca, and Alfonso Ortega. The normalized compression distance is resistant to noise. *IEEE Transactions on Information Theory*, 53(5):1895–1900, 2007.

[7] Rajasekhar Chaganti, Vinayakumar Ravi, and Tuan D. Pham. Image-based malware representation approach with efficientnet convolutional neural networks for effective malware classification. *Journal of Information Security and Applications*, 69:103306, 2022.

[8] Rudi Cilibrasi and Paul Vitanyi. Clustering by compression, 2003.

[9] Rudi Cilibrasi and Paul M.B. Vitányi. Clustering by compression. *CoRR*, cs.CV/0312044, 2003.

[10] Emanuele Cozzi, Pierre-Antoine Vervier, Matteo Dell'Amico, Yun Shen, Leyla Bilge, and Davide Balzarotti. The tangled genealogy of iot malware. In *Annual Computer Security Applications Conference*, ACSAC '20, page 1–16, New York, NY, USA, 2020. Association for Computing Machinery.

[11] Khanh Huu The Dam, Thomas Given-Wilson, and Axel Legay. Unsupervised behavioural mining and clustering for malware family identification. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 374–383, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens. Classification of malware by using structural entropy on convolutional neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[13] Tianxiang He, Chansu Han, Ryoichi Isawa, Takeshi Takahashi, Shuji Kijima, Jun'ichi Takeuchi, and Koji Nakao. A fast algorithm for constructing phylogenetic trees with application to iot malware clustering. In Tom Gedeon, Kok Wai Wong, and Minho Lee, editors, *Neural Information Processing*, pages 766–778, Cham, 2019. Springer International Publishing.

[14] Tianxiang He, Chansu Han, Ryoichi Isawa, Takeshi Takahashi, Shuji Kijima, and Jun'ichi Takeuchi. Scalable and fast algorithm for constructing phylogenetic trees with application to iot malware clustering. *IEEE Access*, 11:8240–8253, 2023.

[15] Tianxiang He, Chansu Han, Takeshi Takahashi, Shuji Kijima, and Jun'ichi Takeuchi. Scalable and fast hierarchical clustering of iot malware using active data selection. In *2021 Sixth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 1–6, 2021.

[16] Thomas Hofmann and Joachim Buhmann. Active data clustering. 01 1997.

[17] Chuleui Hong. A distributed hybrid heuristics of mean field annealing and genetic algorithm for load balancing problem. In Salvatore Greco, Yutaka Hata, Shoji Hirano, Masahiro Inuiguchi, Sadaaki Miyamoto, Hung Son Nguyen, and Roman Słowiński, editors, *Rough Sets and Current Trends in Computing*, pages 726–735, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] Shun-Wen Hsiao, Yeali S. Sun, and Meng Chang Chen. Behavior grouping of android malware family. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016.

[19] Xin Hu, Kang G. Shin, Sandeep Bhatkar, and Kent Griffin. Mutantx-s: Scalable malware clustering based on static features. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 187–198, San Jose, CA, June 2013. USENIX Association.

[20] Sachin Jain and Yogesh Kumar Meena. Byte level n–gram analysis for malware detection. In K. R. Venugopal and L. M. Patnaik, editors, *Computer Networks and Intelligent Computing*, pages 51–59, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[21] Md. Enamul Karim, Andrew Walenstein, Arun Lakhotia, and Laxmi Parida. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1-2):13–23, 2005.

[22] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. The similarity metric. *IEEE Trans. Information Theory*, 50(12):3250–3264, 2004.

[23] P. Minton, H. Raiffa, and Robert Schlaifer. Applied statistical decision theory. *American Mathematical Monthly*, 69:72, 1962.

[24] Lakshmanan Nataraj, Shanmugavadivel Karthikeyan, Grégoire Jacob, and B. S. Manjunath. Malware images: visualization and automatic classification. In *VizSec '11*, 2011.

[25] Takuya Okuyama, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Masanao Yamaoka. Binary optimization by momentum annealing. *Phys. Rev. E*, 100:012111, Jul 2019.

[26] Jonathan Oliver, Muqeet Ali, and Josiah Hagen. Hac-t and fast search for similarity in security. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–7, 2020.

[27] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. IoTPOT: analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[28] Joon-Young Paik, Rize Jin, and Eun-Sun Cho. Malware classification using a byte-granularity feature based on structural entropy. *Computational Intelligence*, 38(4):1536–1558, 2022.

[29] Jan Puzicha, Thomas Hofmann, and Joachim M. Buhmann. A theory of proximity based clustering: structure detection by optimization. *Pattern Recognition*, 33(4):617–634, 2000.

[30] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark Mclean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14, 02 2018.

[31] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19:639–668, 06 2011.

[32] Jorma Rissanen. Modeling by shortest data description. *Autom.*, 14(5):465–471, 1978.

[33] N Saitou and M Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4(4):406–425, 1987.

[34] Shaharuddin Salleh and Albert Y. Zomaya. Multiprocessor scheduling using mean-field annealing. *Future Generation Computer Systems*, 14(5):393–408, 1998. Bio-inspired solutions to parallel processing problems.

[35] David Salomon. *Data compression - The Complete Reference, 4th Edition*. Springer, 2007.

[36] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 230–253, Cham, 2016. Springer International Publishing.

[37] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[38] Claudia Solís-Lemus, Paul Bastide, and Cécile Ané. PhyloNetworks: A Package for Phylogenetic Networks. *Molecular Biology and Evolution*, 34(12):3292–3298, 09 2017.

[39] Sadegh Torabi, Mirabelle Dib, Elias Bou-Harb, Chadi Assi, and Mourad Debbabi. A strings-based similarity analysis approach for characterizing iot malware and inferring their underlying relationships. *IEEE Networking Letters*, 3(3):161–165, 2021.

[40] P. Vinod, V. Laxmi, M. S. Gaur, and Grijesh Chauhan. Momentum: Metamorphic malware exploration techniques using msa signatures. In *2012 International Conference on Innovations in Information Technology (IIT)*, pages 232–237, 2012.

[41] Stephanie Wehner. Analyzing worms and network traffic using compression. *J. Comput. Secur.*, 15(3):303–320, 2007.

[42] Ruiping Yin, Kan Li, Guangquan Zhang, and Jie Lu. Detecting overlapping protein complexes in dynamic protein-protein interaction networks by developing a fuzzy clustering algorithm. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2017.

[43] Takumi Yone. Phylogenetic tree estimation for large-scale malware datasets. Master's thesis, Kyushu University, Japan, 2016. (written in Japanese).

[44] Enmin Zhu, Jianjie Zhang, Jijie Yan, Kongyang Chen, and Chongzhi Gao. N-gram malgan: Evading machine learning detection via feature n-gram. *Digital Communications and Networks*, 8(4):485–491, 2022.

[45] T. Zoller and J.M. Buhmann. Active learning for hierarchical pairwise data clustering. In *Proceedings 15th International Conference on Pattern Recognition. ICPR*, volume 2, pages 186–189 vol.2, 2000.