IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 15 December 2022, accepted 11 January 2023, date of publication 23 January 2023, date of current version 26 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3238711

APPLIED RESEARCH

Scalable and Fast Algorithm for Constructing Phylogenetic Trees With Application to IoT Malware Clustering

TIANXIANG HE[1], CHANSU HAN[2], RYOICHI ISAWA[2], (Member, IEEE), TAKESHI TAKAHASHI[2], (Member, IEEE), SHUJI KIJIMA[1], AND JUN'ICHI TAKEUCHI[1], (Member, IEEE)

[1]Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819-0395, Japan
[2]National Institute of Information and Communications Technology, Koganei 184-8795, Japan

Corresponding author: Tianxiang He (he@me.inf.kyushu-u.ac.jp)

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

ABSTRACT With the development of IoT devices, there is a rapid increase in new types of IoT malware and variants, causing social problems. The malware's phylogenetic tree has been used in many studies for malware clustering or better understanding of malware evolution. However, when dealing with a large-scale malware set, conventional methods for constructing a phylogenetic tree is very time-consuming or even cannot be done in a realistic time. To solve this problem, we propose a high-speed, scalable phylogenetic tree construction algorithm with a clustering algorithm to cluster it. The proposed method involves the following steps: (1) Calculating the similarity of the specimen pairs using the normalized compression distance. (2) Creating a phylogenetic tree containing all specimens, instead of calculating the similarity of all pairs of a specimen, our algorithm only calculates a small part of the similarity matrix. (3) Dividing the phylogenetic tree into clusters by applying the minimum description length criterion. In addition, we propose a new online processing algorithm to add new malware specimens into the existing phylogenetic tree sequentially. Our goal is to reduce the computational cost of constructing the phylogenetic tree and improve the clustering accuracy of our previous research. We evaluated our method's clustering accuracy and scalability with 65,494 IoT malware specimens. The results showed that our algorithm reduced the computation by 97.52% compared with the conventional method. Our clustering algorithm achieved accuracies of 95.5% and 99.3% for clustering family name and architecture name, respectively.

INDEX TERMS Clustering, IoT malware, MDL criterion, phylogenetic tree.

I. INTRODUCTION

IoT malware has seen 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 constructing a phylogenetic tree for malware clustering. 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 for constructing a phylogenetic tree are very time-consuming when facing a large malware set. Therefore, Our goal is to achieve automatic clustering from a large-scale malware specimen set by constructing a

The associate editor coordinating the review of this manuscript and approving it for publication was Easter Selvan Suviseshamuthu.

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

phylogenetic tree. Our method is outlined as follows: we first calculate the distances between malware specimens and then use these distances to create a phylogenetic tree. Finally, we divide the phylogenetic tree into clusters.

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], [8]. 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 for creating a phylogenetic tree is the neighbor-joining method [26]. The neighbor-joining method takes a distance matrix as an input and outputs a phylogenetic tree with tree distance approximating the input distance matrix. However, the neighbor-joining method requires calculating all pairs of distances, which means $(N^2+N)/2$ times of compression attempts in our case. Here, N is the number of data samples. The computation time of the NCD matrix is problematic when N is large. Therefore, our previous work proposed a scalable method for clustering malware by constructing a phylogenetic tree [12].[2] Our previous method consisted of a fast phylogenetic tree construction algorithm and a clustering algorithm. 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. Clustering was achieved by appropriately dividing the phylogenetic tree into subtrees. We applied our method and constructed a phylogenetic tree of 4,109 IoT malware specimens in [12].

But when we applied our previous method to a much larger, multi-architecture malware set, the clustering accuracy dropped significantly. Hence, in this paper, we applied a new clustering algorithm: the minimum description length (MDL) criterion as the division criterion [25]. Based on the information criterion, the `MDL Criterion` decides whether to divide a cluster (subtree) into smaller clusters. We not only improved the clustering accuracy but also used a much larger dataset containing 65,494 malware specimens to evaluate our method's scalability. 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 phylogenetic tree without fully reconstructing it.

Our contributions are threefold:

- We present a scalable and fast algorithm to construct phylogenetic trees, which significantly reduces the computational cost.
- We present a scalable clustering algorithm applying the `MDL Criterion`. 95.5% and 99.3% accuracies of clustering family name and architecture name were achieved, respectively.
- We present an online processing algorithm that successively adds new specimens into the phylogenetic tree while maintaining the clustering accuracy.

The rest of this paper is organized as follows. The second section introduces the related studies. The third section introduces existing methods we applied in our study. The fourth section presents our newly proposed algorithm, i.e., clustering algorithm and online processing algorithm. In the fifth section, we discuss the application of our algorithm with 65,494 specimens of IoT malware. In particular, we show how much we reduce the computational cost and the extent to which we improve the clustering accuracy. Finally, the last section discusses the limitations and directions for future research.

II. RELATED WORK

A. BINARY-LEVEL STATIC ANALYSIS

We applied NCD for feature extraction that directly calculates the similarities between malware's byte sequences. Details about NCD will be introduced in subsection III-A. Other binary-level feature extraction methods include N-grams [16], [23], [35], entropy-based [11], [22], and image-based techniques [5], [7], [20], 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 [16]. 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 that Markov chain algorithm can match more same patterns in the bytes 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 bytes 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

[2] This paper was partly presented at the International Conference on Neural Information Processing (ICONIP), 2019 [12]. The clustering algorithm and experiments were updated.

TABLE 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 [17]	Static analysis	N-gram and N-perm	170	UPGMA	No
Vinod et al. 2012 [33]	Static analysis	NCD	790	Not mentioned	No
Hsiao et al. 2016 [14]	Dynamic analysis	system call	1,200	The unweighted pair group method	No
Wehner 2007 [32]	Static analysis	IDA	1,200	The neighbor-joining method	No
Bailey et al. 2007 [3]	Dynamic analysis	Behaviors	3,700	The shortest distance method	No
Cozzi et al. 2020 [9]	Both dynamic and static analysis	Code-based	36,574	HNSW + the minimum spanning tree	Yes

TABLE 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 [10]	Static analysis	System call graph	21,000	graph clustering	Yes
Ricck et al. 2011 [24]	Dynamic analysis	Behavior	33,698	Prototype-based clustering	Yes
Torabi et al. 2021 [31]	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 [15]	Dynamic analysis	Code instruction	130,000	Prototype-Based Clustering	Yes
Oliver et al. 2020 [21]	Static analysis	3-grams and TLSH	10 million	HAC-T	Yes

robustness [1]. Our method and N-gram match exactly the same pattern between byte sequences, while Neural Networks can deal with minor alterations.

B. 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 study constructing a phylogenetic tree is shown in Table 1.

Karim et al. [17] proposed a method to calculate the distance between all malware pairs using n-perms and created a phylogenetic tree. Wehner [33] 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 [14], 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. Vinod et al. [32] 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 previous 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, [9] 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.

C. 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.

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.

Oliver et al. [21] 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. [31] 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 [34] algorithm to perform clustering analysis. Rieck et al. [24] and Hu et al. [15] 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. [10] also took a very different approach: they use IDA pro to extract system call graph for each malware and transfer graphs into vectors applying LSTM.

III. PRELIMINARIES

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

A. 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 [19]. 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.

B. NEIGHBOR-JOINING METHOD

The neighbor-joining method [26] 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 d'_{ij}, defined below, join a newly created node p.

$$d'_{ij} = d_{ij} - \frac{\sum_{k \in L}(d_{ik} + d_{jk})}{|L| - 2},$$

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 $T$
while $|L| \geq 2$ do
    choose $(u, v)$ $(u \neq v)$ which minimizes $d'_{u,v}$
    create a new node $p$, $V := V \cup \{u, v, p\}$, $E = E \cup \{\{u, p\}, \{v, p\}\}$
    branch length of$\{u, p\}$ is defined as: $D_{u,p} = (d_{u,v} + \frac{\sum_{k \in L\setminus\{u,v\}}(d_{uk} - d_{vk})}{|L|-2})/2, L := (L \setminus \{u, v\}) \cup \{p\}$
    branch length of $\{v, p\}$ is defined as: $D_{v,p} = (d_{u,v} + \frac{\sum_{k \in L\setminus\{u,v\}}(d_{vk} - d_{uk})}{|L|-2})/2$
    for $w \in L \setminus \{p\}$, $d(w, p) := (1/2)(d_{u,w} + d_{v,w} - d_{u,v})$
end while
return $T = (V, E), D$
```

IV. PROPOSED METHOD

In this section, we introduce the fast algorithm for constructing a phylogenetic tree, the new clustering algorithm applying the `MDL Criterion` and the online processing algorithm.

A. FAST ALGORITHM FOR CONSTRUCTING A PHYLOGENETIC TREE

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.

The algorithm is outlined as follows. The schematic diagram and the flow chart are shown in Fig. 1 and Fig. 2, respectively. First, the algorithm randomly selects a seeds set $S \subset L$ $(|S| = k, |L| = N)$ with k $(k \ll N)$. 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.

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 2, where ∂T denotes a set of all leaves of T.

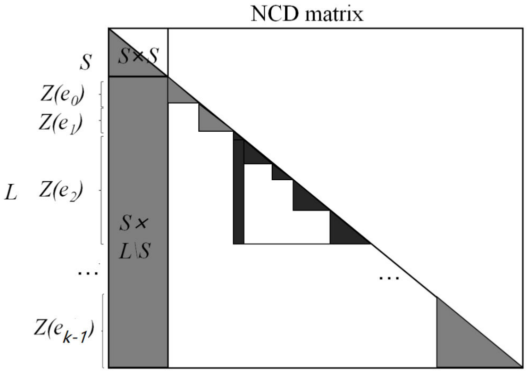


FIGURE 1. The schematic diagram of the phylogenetic tree construction algorithm.

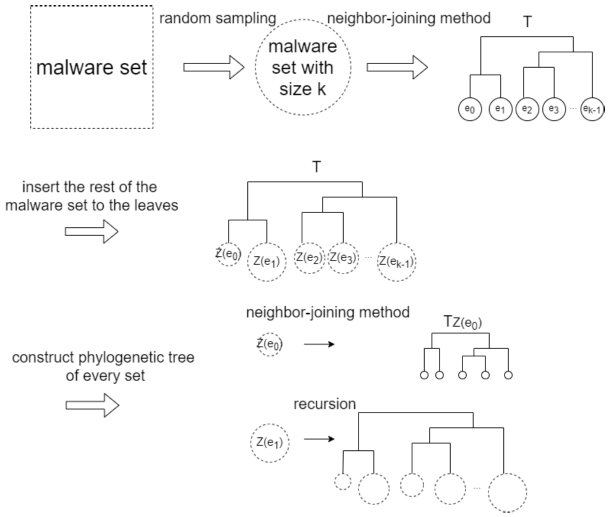


FIGURE 2. The flow chart of the phylogenetic tree construction algorithm.

Algorithm 2 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 2 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. 1 are calculated in Algorithm 2. As a randomized algorithm, Algorithm 2'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 $O(N \log N)$. Details of the computational cost are presented in the appendix.

B. CLUSTERING ALGORITHM

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 our previous 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` [25].

The `MDL Criterion` is a model selection criterion based on information theory [25]. `MDL Criterion` is defined as below:

$$\mathrm{MDL} = -2 \log L(\hat{\theta}; x) + m_j \log(n). \tag{1}$$

where L is the likelihood function, θ is the maximum likelihood estimate, m_j is the number of dimensions of θ, 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.

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

Here, L is the likelihood function, θ 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, 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

Algorithm 3 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
    $Tc = Tc \cup C$
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
$Tc = \emptyset$
$Tc$ = Cluster($T$)
for $T_i \in Tc$ 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
                $Tc$= $Tc - T_i - Tn$
                $Tc = Tc \cup C$
                break
            else
                $Tn$=$T_i$'s next nearest cluster
                count++1
            end if
        end while
    end if
end for
```

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. The description length of the divided cluster is calculated by:

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

Here, x_k is the malware sample contained in the cluster C_k, and n_k is its number.

Algorithm 4 Algorithm for Online Processing

```
Require: Phylogenetic tree $T$, clustering result $Tc$, new malware set $L$
Ensure: New phylogenetic tree $T'$
$S = \emptyset$
for each $T_i \in Tc$ do
    $S = S \cup \{e\}$, where $e$ is the central malware of cluster $T_i$
end for
calculate the distances $d_{ij}$ for $(i, j) \in S \times L$
for each malware $l \in L$ do
    insert $l$ into nearest cluster
end for
for each $T_i \in Tc$ do
    re-construct the phylogenetic tree of $T_i$
end for
join all clusters into new phylogenetic tree $T'$
```

If DL(C_1, C_2, C_3) > DL(C), the cluster before the division is considered better, and the division is rejected. If DL(C_1, C_2, C_3) < 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 DL(C) of the cluster after merging is smaller than the description length DL(C_1, C_2) of the clusters before merging. The pseudocode of the algorithm is shown in Algorithm 3.

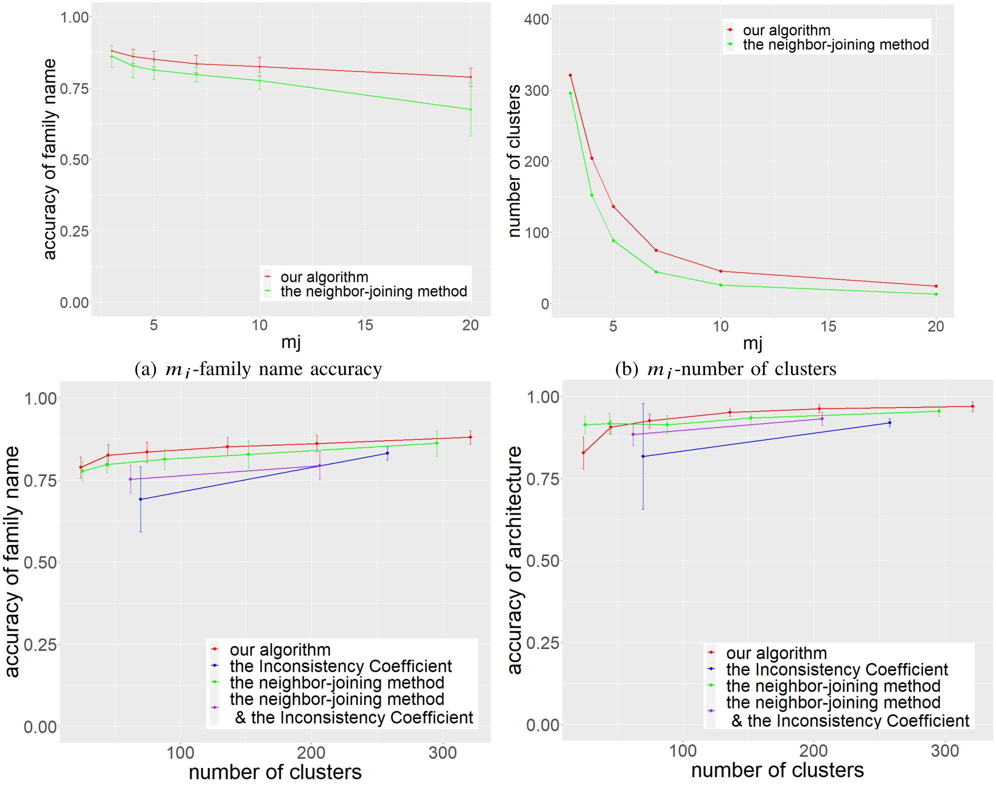
C. ONLINE PROCESSING ALGORITHM

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 phylogenetic tree.

After creating a phylogenetic tree and clustering it, we calculate the distance between 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. Finally,

TABLE 3. 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



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

all clusters are joined back to a phylogenetic tree. The pseudocode of the algorithm is shown in Algorithm 4.

The evaluation of the proposed algorithm is introduced in the next section.

V. EXPERIMENT AND RESULTS

In this section, we evaluate our algorithm with 64,494 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`, and 3) how well our online processing algorithm reduces the computation time while maintaining the clustering accuracy.

A. DATASET

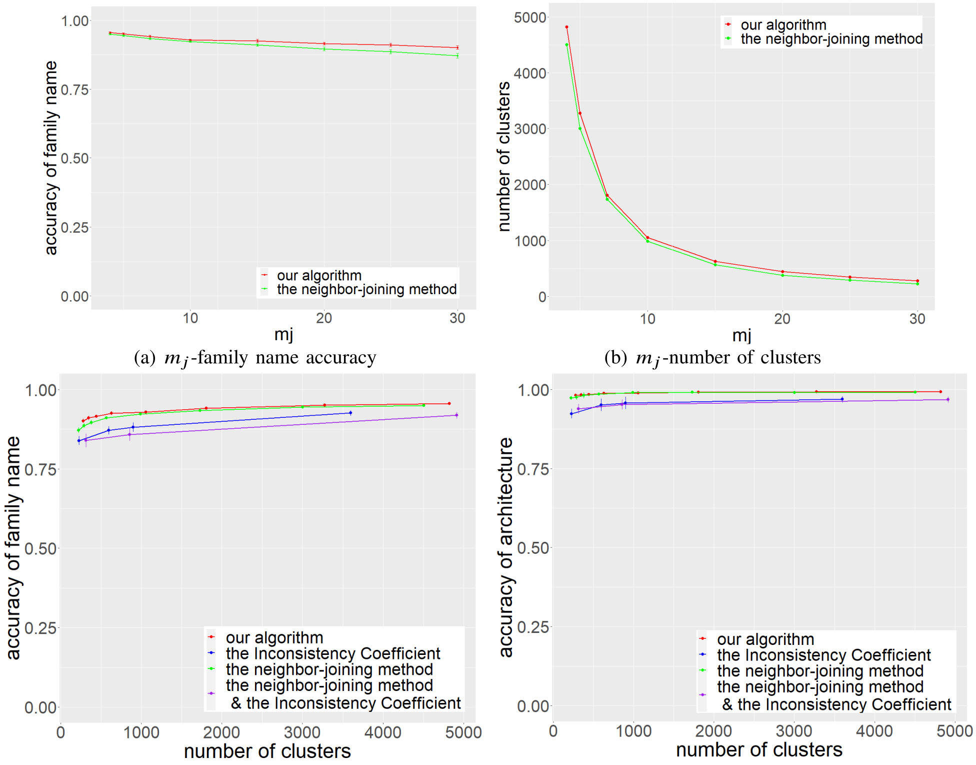
We collected 65,494 Linux malware specimens (mostly IoT malware) from VirusTotal, from November 2018 to February 2019. The breakdown of the dataset is shown in Table 3. The malware family name was decided by AVClass [28]. AVClass is implemented as a Python tool to label malware samples using VirusTotal JSON reports as input.

B. EXPERIMENTAL SETUP

In this subsection, we introduce the detailed setup and evaluation metrics of our experiments.

1) SETUP

For performance comparison, we constructed phylogenetic trees with both the our fast algorithm and the neighbor-joining



(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 4. **Experimental result and comparison between our algorithm, the neighbor-joining method and the** `Inconsistency Coefficient`.

method. We then clustered them with both the `MDL Criterion` and the `Inconsistency Coefficient`, which is the method we applied previously. 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 similar number of specimens as in our previous study: 4,000 specimens, which are randomly selected from the malware set. Second, we evaluated our algorithm with 65,494 specimens.

In the online processing experiment, we compared it to batch processing. The results show that we saved the time of reconstructing the phylogenetic tree when new specimens are added to the dataset while maintaining the same level of clustering accuracy.

2) PARAMETER TUNING

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.

3) IMPLEMENTATION

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) [27] 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 III-B. 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 texttttJulia programming language as an alternative [29].

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.

4) 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:

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

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

TABLE 4. Confusing matrix of test malware specimens.

		Actual family																				
		Mirai	Bashlite	Tsunami	Mobidash	Others	Wroba	Dofloo	Setag	Fakebank	Ddostf	Locker	Xorddos	Lotoor	Hajime	Hiddad	Dnsamp	Xmrig	Feejar	Fiforeg	sshdoor	Rootnik
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

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 cluster's center specimen. The cluster's family name is decided by the majority specimen's family name in that cluster.

C. EVALUATION RESULTS

1) SMALL-SCALE EXPERIMENT 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. 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(a) and Fig. 3(b), the horizontal axis is the parameter m_j introduced in subsection IV-B 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. 4(c) and Fig. 4(d) to show the difference better. However, our algorithm still achieves a slightly higher accuracy.

Fig. 3(c) and Fig. 3(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.
- 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 our previous work. 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.

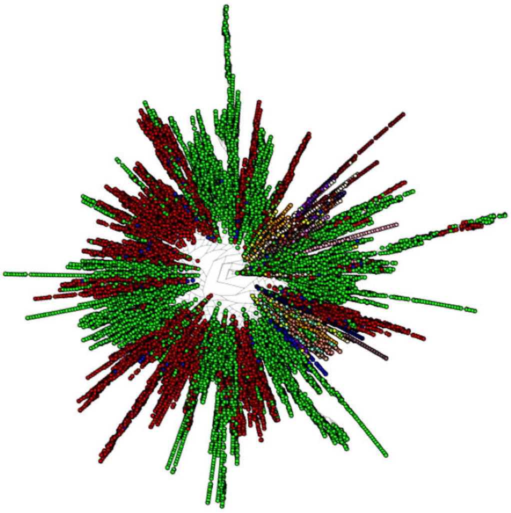


FIGURE 5. Phylogenetic tree constructed using our algorithm.

2) LARGE-SCALE EXPERIMENT WITH 65,494 SPECIMENS

The phylogenetic tree constructed using 65,494 specimens is shown in Fig. 5, 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 [29], 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.

It can be seen from Fig. 4(c) and Fig. 4(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 Table. 4. 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. Table. 4 shows that we achieve high accuracy for most minor families.

3) EXPERIMENT OF ONLINE PROCESSING ALGORITHM

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.

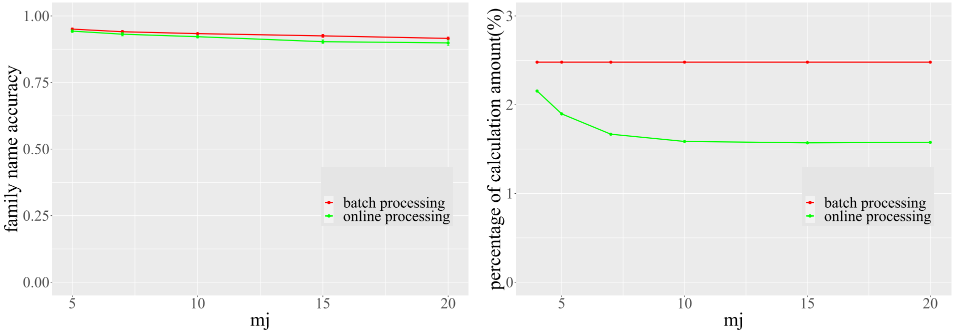
Fig. 6(a) shows that our online processing algorithm achieved close accuracy compared to batch processing, the clustering accuracy was only reduced by about 1%. Fig. 6(b) shows that our online processing algorithm not only saved the time of re-construct a new phylogenetic tree but also reduced the computational 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. 6(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. 8(a).
- 2) 10000 and 2000, the experiment result is shown in Fig. 8(b).



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

FIGURE 6. Results of online processing algorithm compared with batch processing.

- 3) 20000 and 4000, the experiment result is shown in Fig. 8(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.

In the experiment 2, we investigated how the properties of the new specimens affects the online processing algorithm's accuracy. Fig. 7 shows the average distance among specimens in different periods. We can see from Fig. 7 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. 9: 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.

VI. DISCUSSION

A. PERFORMANCE GUARANTEE

Our phylogenetic tree construction algorithm is a randomized algorithm; therefore, there is no performance guarantee. However, we measured the standard deviation of the clustering accuracy of the 10-fold cross-validation in Fig. 4. As shown in the figure, the standard deviations of the clustering accuracy are small. For instance, the standard deviation is 0.33% when $m_j = 7$, and the accuracy of the malware family name is 94.07%. Therefore, it is reasonable to believe that the randomness does not significantly affect the clustering result.

B. LIMITATION

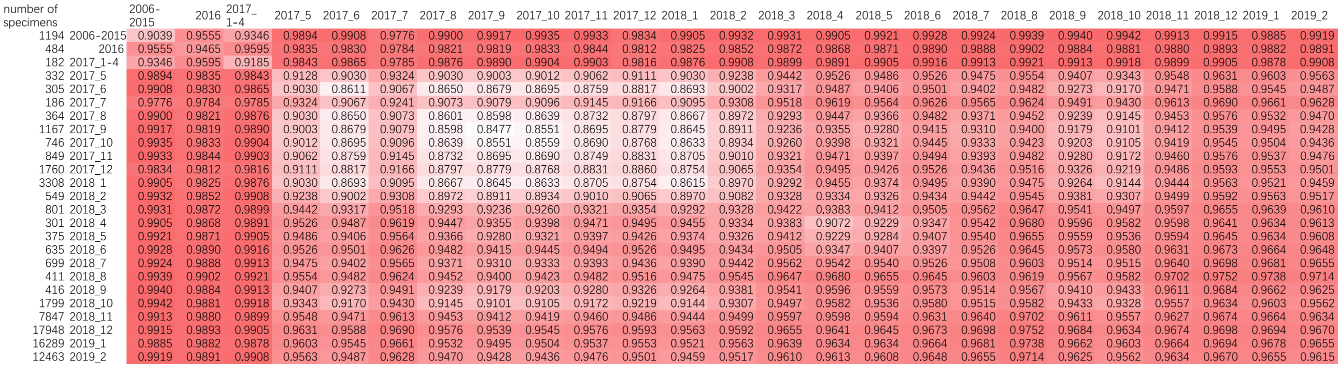
Since the similarity of malware specimens is measured by NCD, it depends on the binary similarity between the two specimens. Thus, encrypted or packed malware whose binary has been significantly changed is difficult to cluster. Although unpack tools can solve this problem to some extent, we currently cannot deal with those packed malware that cannot be unpacked.

C. NCD's ADVANTAGES

We applied NCD for feature extraction that directly calculates the similarities between malware's binary code. Compared to other Sota feature extraction methods, our method does not need the dynamic analysis environment or the time to analyze each malware. Sota feature extraction method using dynamic analysis is effective but it takes time to collect a large amount of data because the malware must be executed at least once before the logs can be collected. Moreover, static analysis using the disassembler tool IDA pro is also a Sota feature extraction method. However, it also takes a long time to disassemble all the malware. Another advantage of the NCD approach is its flexibility. It does not require any domain knowledge and can apply to many other data formats, such as text, video, and music, without requiring re-designed.

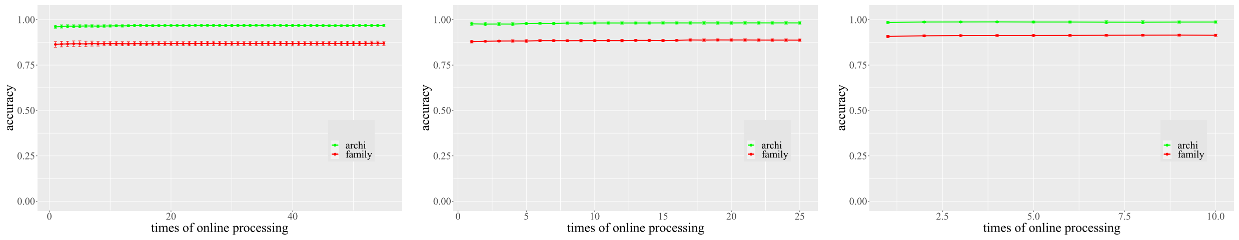
D. PHYLOGENETIC TREE's ADVANTAGES

Our method created the largest phylogenetic tree and achieved a high clustering accuracy at about 94% to 95%.



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 7. Average distance among specimens in different period.



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

FIGURE 8. Results of online processing experiment 1.

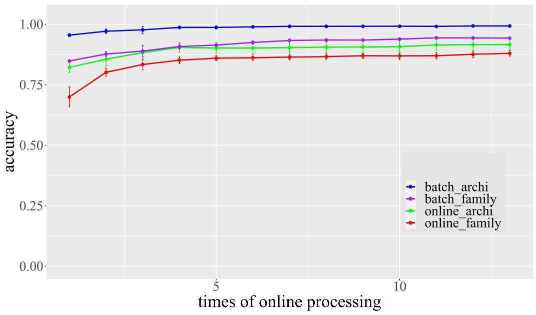


FIGURE 9. Result of online processing experiment 2.

But if only regarding the clustering scalability and regardless of the phylogenetic tree construction, many studies have achieved better scalability. Compared to these studies, the advantage of constructing a malware's phylogenetic tree is that clustering using a phylogenetic tree can provide more information about the malware's evolution path, which is connected to our future work.

E. FUTURE WORKS

For our future works, we are attempting to build a hybrid platform that integrates various cyber threat analysis methods [30]. As part of this project, we are working on the challenge of static analysis of malware.

Kawasoe et al. [18] have proposed investigating malware specimen functions by generating Function Call Sequence Graphs (FCSG). Our future work aims to combine the method of Kawasoe et al. and the clustering results of this study to capture the functionality transition and evolution of malware variants.

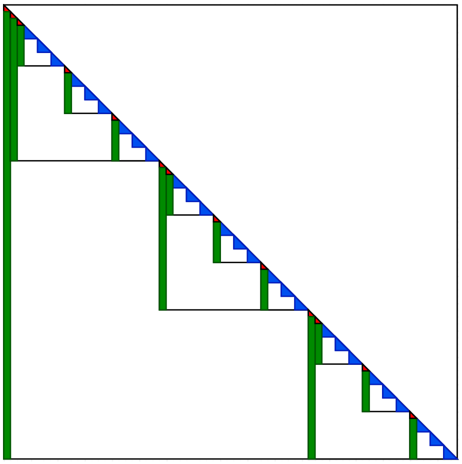


FIGURE 10. The schematic diagram of the computation cost of our algorithm.

As another future work, we are using information theory to choose the seed smartly in the recursion part of the phylogenetic tree construction to reach a better computation reduction rate.

Moreover, we are also using active learning to develop a new algorithm for constructing a phylogenetic tree [13]. The active learning algorithm allows us to smartly select the most informative part of the distance matrix to calculate. We expect this method would achieve better scalability.

VII. CONCLUSION

In this paper, we proposed a scalable, efficient, and automatic method for large-scale malware clustering. We presented a fast algorithm for constructing the phylogenetic tree and an algorithm applying the `MDL Criterion` to clustering it. The computational cost of the fast algorithm is O($N\log N$), which significantly reduced the computational cost than the conventional method. 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 our previous method. 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. The O($N\log N$) complexity makes our algorithm scalable for large-scale malware sets. To the best of our knowledge, our work constructed the largest malware phylogenetic tree. Moreover, it was previously impossible to construct a malware phylogenetic tree at our size in a realistic time. Now, we can construct and cluster it in real-time.

APPENDIX. COMPUTATION COST

In this appendix section, we will show the detailed explanation of how our algorithm reduces the computation cost. 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%.

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.10, and it is calculated as:

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

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

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

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

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

ACKNOWLEDGMENT

The authors are grateful to Kohei Kubo, Takumi Yone, Yukiko Yamauchi and Masafumi Yamashita for discussions about the algorithm for estimating a phylogenetic tree of big data.

REFERENCES

[1] A. Abusitta, M. Q. Li, and B. C. M. Fung, "Malware classification and composition analysis: A survey of recent developments," *J. Inf. Secur. Appl.*, vol. 59, Jun. 2021, Art. no. 102828. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212621000648

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, and Z. Durumeri, "Understanding the Mirai botnet," in *Proc. 26th USENIX Conf. Secur. Symp.*, 2017, pp. 1093–1110.

[3] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *Proc. 10th Int. Symp. RAID*, Gold Coast, QLD, Australia, Sep. 2007, pp. 178–197.

[4] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2009, pp. 8–11.

[5] N. Bhodia, P. Prajapati, F. Di Troia, and M. Stamp, "Transfer learning for image-based malware classification," 2019, *arXiv:1903.11551*.

[6] M. Cebrian, M. Alfonseca, and A. Ortega, "The normalized compression distance is resistant to noise," *IEEE Trans. Inf. Theory*, vol. 53, no. 5, pp. 1895–1900, May 2007.

[7] R. Chaganti, V. Ravi, and T. D. Pham, "Image-based malware representation approach with EfficientNet convolutional neural networks for effective malware classification," *J. Inf. Secur. Appl.*, vol. 69, Sep. 2022, Art. no. 103306. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212622001570

[8] R. Cilibrasi and P. M. Vitányi, "Clustering by compression," 2003, *arXiv:0312044*.

[9] E. Cozzi, P.-A. Vervier, M. Dell'Amico, Y. Shen, L. Bilge, and D. Balzarotti, "The tangled genealogy of IoT malware," in *Proc. Annu. Comput. Secur. Appl. Conf.*, New York, NY, USA, Dec. 2020, pp. 1–16.

[10] K. H. T. Dam, T. Given-Wilson, and A. Legay, "Unsupervised behavioural mining and clustering for malware family identification," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, New York, NY, USA, Mar. 2021, pp. 374–383, doi: 10.1145/3412841.3441919.

[11] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks," in *Proc. 32nd AAAI Conf. Artif. Intell. 30th Innov. Appl. Artif. Intell. Conf. 8th AAAI Symp. Educ. Adv. Artif. Intell.*, 2018, pp. 1–6.

[12] T. He, C. Han, R. Isawa, T. Takahashi, S. Kijima, J. Takeuchi, and K. Nakao, "A fast algorithm for constructing phylogenetic trees with application to IoT malware clustering," in *Neural Information Processing*, T. Gedeon, K. W. Wong, and M. Lee, Eds. Cham, Switzerland: Springer, 2019, pp. 766–778.

[13] T. He, C. Han, T. Takahashi, S. Kijima, and J. Takeuchi, "Scalable and fast hierarchical clustering of IoT malware using active data selection," in *Proc. 6th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Dec. 2021, pp. 1–6.

[14] S.-W. Hsiao, Y. S. Sun, and M. C. Chen, "Behavior grouping of Android malware family," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[15] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "MutantX-S: Scalable malware clustering based on static features," in *Proc. USENIX Annu. Tech. Conf.* San Jose, CA, USA: USENIX Assoc., Jun. 2013, pp. 187–198.

[16] S. Jain and Y. K. Meena, "Byte level *n*–gram analysis for malware detection," in *Computer Networks and Intelligent Computing*, K. R. Venugopal and L. M. Patnaik, Eds. Berlin, Germany: Springer, 2011, pp. 51–59.

[17] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," *J. Comput. Virol.*, vol. 1, nos. 1–2, pp. 13–23, 2005.

[18] R. Kawasoe, C. Han, R. Isawa, T. Takahashi, and J. Takeuchi, "Investigating behavioral differences between IoT malware via function call sequence graphs," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, New York, NY, USA, Mar. 2021, pp. 1674–1682, doi: 10.1145/3412841.3442041.

[19] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitányi, "The similarity metric," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 3250–3264, Dec. 2004.

[20] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. VizSec*, 2011, pp. 1–7.

[21] J. Oliver, M. Ali, and J. Hagen, "HAC-T and fast search for similarity in security," in *Proc. Int. Conf. Omni-Layer Intell. Syst. (COINS)*, Aug. 2020, pp. 1–7.

[22] J. Paik, R. Jin, and E. Cho, "Malware classification using a byte-granularity feature based on structural entropy," *Comput. Intell.*, vol. 38, no. 4, pp. 1536–1558, Aug. 2022, doi: 10.1111/COIN.12521.

[23] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte *n*-gram features for malware classification," *J. Comput. Virol. Hacking Techn.*, vol. 14, no. 1, pp. 1–20, Feb. 2018.

[24] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *J. Comput. Secur.*, vol. 19, no. 4, pp. 639–668, Jun. 2011.

[25] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978, doi: 10.1016/0005-1098(78)90005-5.

[26] N. Saitou and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetic trees," *Mol. Biol. Evol.*, vol. 4, no. 4, pp. 406–425, Jul. 1987.

[27] D. Salomon, *Data Compression: The Complete Reference*, 4th ed. Cham, Switzerland: Springer, 2007.

[28] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVCLASS: A tool for massive malware labeling," in *Research in Attacks, Intrusions, and Defenses*, F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, Eds. Cham, Switzerland: Springer, 2016, pp. 230–253.

[29] C. Solís-Lemus, P. Bastide, and C. Ané, "PhyloNetworks: A package for phylogenetic networks," *Mol. Biol. Evol.*, vol. 34, no. 12, pp. 3292–3298, Dec. 2017, doi: 10.1093/MOLBEV/MSX235.

[30] T. Takahashi, Y. Umemura, C. Han, T. Ban, K. Furumoto, O. Nakamura, K. Yoshioka, J. Takeuchi, N. Murata, and Y. Shiraishi, "Designing comprehensive cyber threat analysis platform: Can we orchestrate analysis engines?" in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Affiliated Events*, Mar. 2021, pp. 376–379.

[31] S. Torabi, M. Dib, E. Bou-Harb, C. Assi, and M. Debbabi, "A strings-based similarity analysis approach for characterizing IoT malware and inferring their underlying relationships," *IEEE Netw. Lett.*, vol. 3, no. 3, pp. 161–165, Sep. 2021.

[32] P. Vinod, V. Laxmi, M. S. Gaur, and G. Chauhan, "MOMENTUM: MetamOrphic malware exploration techniques using MSA signatures," in *Proc. Int. Conf. Innov. Inf. Technol. (IIT)*, Mar. 2012, pp. 232–237.

[33] S. Wehner, "Analyzing worms and network traffic using compression," *J. Comput. Secur.*, vol. 15, no. 3, pp. 303–320, Mar. 2007. [Online]. Available: http://content.iospress.com/articles/journal-of-computer-security/jcs287

[34] R. Yin, K. Li, G. Zhang, and J. Lu, "Detecting overlapping protein complexes in dynamic protein-protein interaction networks by developing a fuzzy clustering algorithm," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jul. 2017, pp. 1–6.

[35] E. Zhu, J. Zhang, J. Yan, K. Chen, and C. Gao, "N-gram MalGAN: Evading machine learning detection via feature *n*-gram," *Digit. Commun. Netw.*, vol. 8, no. 4, pp. 485–491, Aug. 2022.

TIANXIANG HE received the B.Sc. degree in applied mathematics from the Beijing University of Posts and Telecommunications, in 2017, and the M.S. degree in informatics engineering from Kyushu University, in 2020, where he is currently pursuing the Ph.D. degree. His research interest includes malware analysis.

CHANSU HAN received the B.E. degree in computer science and the M.S. and Ph.D. degrees in informatics engineering from Kyushu University, in 2016, 2018, and 2021, respectively. He is currently a Researcher with the National Institute of Information and Communications Technology (NICT), Japan. His research interest includes analyzing and solving problems in the cybersecurity field (especially networks and malware) using machine learning.

RYOICHI ISAWA (Member, IEEE) received the B.E. and M.E. degrees from the University of Tokushima, Japan, in 2004 and 2006, respectively, and the Ph.D. degree from Kobe University, Japan, in 2012. He is currently a Senior Researcher with the National Institute of Information and Communications Technology (NICT), Japan. His current research interests include malware analysis, network security, and hardware security.

TAKESHI TAKAHASHI (Member, IEEE) received the Ph.D. degree in telecommunications from Waseda University, in 2005. He was a Researcher at the Tampere University of Technology, from 2002 to 2004; a JSPS Research Fellow at Waseda University, from 2004 to 2006; and a Business Consultant at Roland Berger Ltd., from 2006 to 2009. Since 2009, he has been with the National Institute of Information and Communications Technology, where he is currently an Associate Director. Further, he was a Visiting Research Scholar at the University of California, Santa Barbara, Santa Barbara, CA, USA, from 2019 to 2020. His research interests include cybersecurity and machine learning.

SHUJI KIJIMA received the Ph.D. degree in mathematical informatics from The University of Tokyo, in 2007. He is currently an Associate Professor with Kyushu University. His research interests include analysis of algorithms and discrete mathematics.

JUN'ICHI TAKEUCHI (Member, IEEE) received the B.Sc. degree in physics and the Dr.Eng. degree in mathematical engineering from The University of Tokyo, in 1989 and 1996, respectively. From 1996 to 1997, he was a Visiting Research Scholar with the Department of Statistics, Yale University, New Haven, CT, USA. From 1989 to 2006, he worked with NEC Corporation, Japan. In 2006, he joined Kyushu University, Fukuoka, Japan, where he is currently a Professor of mathematical engineering. His research interests include mathematical statistics, information geometry, information theory, data science, and machine learning. He is a member of IEICE and JSIAM.