Fast Clustering of High Dimensional Data Clustering the Malware Bazaar Dataset

Jonathan Oliver
TrendMicro
jon_oliver@trendmicro.com

Muqeet Ali
TrendMicro
muqeet_ali@trendmicro.com

Haoping Liu
TrendMicro
haoping_liu@trendmicro.com

Josiah Hagen
TrendMicro
josiah_hagen@trendmicro.com

Abstract

We consider the situation where we wish to cluster some data according to a distance function. The data under consideration is high dimensional, and the number of clusters is unknown. We propose a single-linkage algorithm which has features suitable for clustering large datasets of this type. We define a clustering of a data set as being optimal with respect to a distance parameter, $CDist$, if every pair of points which occur within a distance of $CDist$ are in the same cluster. The proposed algorithm uses Vantage-Point-Trees to provide fast nearest neighbor search resulting in optimal single-linkage clustering with time complexity $O(Nlog^2N)$. We offer adjustments to the algorithm that reduce the time complexity to $O(NlogN)$, but we lose the guarantee of optimal clustering. We argue that features of high dimensional spaces mean that the modification is likely to result in near optimal single-linkage clustering. We then explore a public data corpus, and verify that the proposed algorithm results in near optimal clustering with time complexity $O(NlogN)$.

1 Introduction

We consider the situation where we wish to cluster some data according to a distance function. We were motivated by the requirements from computer security applications and found that existing clustering algorithms had limitations. The requirements were to cluster data large sets of high dimensional data. The number of clusters is unknown, potentially the size of important clusters could be very small (as small as 2), and the density of clusters varies considerably.

There is a number of clustering approaches to consider: Kmeans and Kmedoid require the number of clusters be defined before analysis and they are unreliable for identifying small clusters. DBScan [7, 22] will exhibit $O(N^2)$ time complexity with high dimensional data, and setting its minPts parameter (which defines a minimum density) is difficult when we expect the densities of clusters to vary significantly. Naive hierarchical clustering (or single link clustering such as SLINK [23]) resolve the issues around the number and density of clusters, but their $O(N^2)$ time complexity makes their use infeasible.

There are also clustering approaches using Locality Sensitive Hashes (LSH) [8]. LSH have received considerable attention in technical papers [12, 8, 6]. The LSH-based clustering approaches fall into 2 categories:

1. Collision-based methods that use the LSH to map points onto a bucket in constant time and thereby achieve $O(N)$ clustering (such as [13]). If the collision rate is not adequate or false alarms are occurring, then one approach is to use multiple LSH functions [15].
2. Distance-based methods that use the LSH to define the distance metric between points, and use indexing structures such as Vantage-Point-Trees (VPT) [25, 26] to do fast nearest neighbor search (such as HAC-T [18]).

Collision-based methods are ideal when the LSH buckets have a good fit with the desired clusters. However, it has been reported that creating this good fit can be difficult [24, 11][1]. One area is interesting where there has been much public discussion, evaluation of collision-based versus distance-based schemes is computer security (where the area is described as "fuzzy hashing") [16]. 15-20 years ago there were many collision-based fuzzy hashing schemes for computer security, and this has moved to scoring schemes [16]. In Table 4 of a 2021 comprehensive survey of fuzzy hashing schemes [16], 13 of 14 schemes used distance or similarity scores.

Here, we look at the distance-based LSH clustering [2]. HAC-T does a nearest neighbor search using a VPT for each point in a dataset [18]. If the VPT search can achieve an average time complexity of $O(logN)$ [26], then the overall clustering is time complexity $O(NlogN)$. In this approach, we can use LSH where the buckets are more fine grain than the desired cluster size, but it is not useful to use LSH where the buckets are larger than the desired cluster size.

In this paper, we review the HAC-T algorithm and compare with previous clustering algorithms. The previous description of HAC-T was limited to applications in computer security, and assumed a distance function which only took integer values.

Both HAC-T and DBSCAN use a parameter which defines the distance between points where we will identify points as belonging to the same cluster (ϵ for DBSCAN, $CDist$ for HAC-T). We define a clustering of a data set as being optimal with respect to a distance parameter, $CDist$, if every pair of points which occur within a distance of $CDist$ are in the same cluster. DBSCAN has additional parameter for the minimum number of points to be included in a cluster. DBSCAN can form 'optimal' clusterings in the sense being used here when the minimum number of points in a cluster is 2, but this can result in inefficient time performance.

We adapt HAC-T to the general clustering setting and demonstrate that VPT search can achieve optimal single-linkage clustering. We apply HAC-T to a public corpus, and verify both the quality and performance of the clustering. The code has been open sourced at https://github.com/trendmicro/tlsh/tree/master/tlshCluster .

1.1 Overview of Indexing Methods

There are a number of methods for efficient nearest neighbor search in metric spaces, including K-d trees [5], R-Trees [10] and Vantage-Point-trees [25, 26] (also called metric trees). These methods avoid brute force search by identifying parts of the tree where near neighbors cannot exist, and hence search in those parts of the tree can be abandoned. These methods are also suitable for approximate nearest-neighbor search [17]. And these can be used in clustering applications. For example, DBSCAN [7] uses R-Trees to find all close members of a point under consideration (and thus avoids a linear search through the database).

Vantage-Point-trees (VPT) have been used in fewer applications than K-d trees and R-Trees, which have been in the areas such as image segmentation [14]. VPT are most advantageous for high dimensional data [26], where K-d trees and R-trees are known to degrade [22]. Recently, VPT have proven themselves useful in area of computer security [18, 2].

1.1.1 Vantage Point Tree Search

To enable efficient nearest neighbor search through our dataset D, we build a binary search tree using randomized selection of items to place points at internal nodes. Each internal node of the binary tree contains a "VantagePoint" (from D) and left and right children (LC and RC). In addition, each internal node contains a Threshold value selected such that:

[1]This is an issue shared with DBScan which suffers when the data has clusters with a wide range of densities.
[2]The technique also applies to high dimensional data which has a distance metric defined.

Algorithm 1 $VPTSearch(node, searchItem, notInC, best)$

```
if $node == NULL$ then
    return
end if
$d \leftarrow distMetric(node.Point, searchItem)$
if $(cluster[node.Point] \neq notInC)$ AND $(d < best.Dist)$ then
    $best.Dist \leftarrow d$
    $best.Point \leftarrow node.Point$
end if
if $d \leq node.threshold$ then
    $VPTSearch(node.LC, searchItem, ignoreList, best)$
    if $d + best.Dist \geq node.Threshold$ then
        $VPTSearch(node.RC, searchItem, ignoreList, best)$
    end if
else
    $VPTSearch(node.RC, searchItem, ignoreList, best)$
    if $d - best.Dist \leq node.Threshold$ then
        $VPTSearch(node.LC, searchItem, ignoreList, best)$
    end if
end if
```

- all x in the left subtree satisfy $distMetric(x, VantagePoint) < Threshold$, and
- all x in the right subtree satisfy $distMetric(x, VantagePoint) \geq Threshold.$

Let the root of the tree be $rootVPT$. The procedure for finding a $searchItem$ in a VPT with root $node$ is shown in Algorithm 1. We have adapted the standard VPT search to optionally not return the nearest neighbor which is in a specified cluster ($notInC$). This pseudo-code is provided as python code at https://github.com/trendmicro/tlsh/tree/master/tlshCluster . We note that the distance function should be a distance metric[3], due to the use of the triangle inequality to determine when backtrack is needed. The worst case search time of a VPT is $O(N)$, however in appropriate conditions (high dimensional situations with a reasonable distance metric) the search time is found to be $O(logN)$ (Yianilos [26] argues without proof that the expected search time is $O(logN)$).

1.2 HAC-T

In this section, we present the HAC-T for clustering a dataset D with a distance metric $distMetric$. HAC-T requires the cluster distance parameter, $CDist$, the distance for a single link. This algorithm is optimal w.r.t creating a single linkage clustering. If two points have a distance $\leq CDist$, then either they will be merged in Step 2 (the normal case) or the loop in Step 3 will identify those points and merge their clusters.

1.2.1 Time Complexity

Step 1 is time complexity $O(NlogN)$. Step 2 is also time complexity $O(NlogN)$ as the while loop is executed a maximum of N times, and the deleting an element from a heap takes $O(logN)$ operations. Step 3 consists of the following embedded loops.

```
while $len(Clusters\_to\_examine) > 0$
        for each cluster $C \in Clusters\_to\_examine$
                for each item $A \in C$
                        VPTSearch(rootVPT, A, listMembers(Cluster(C)), best)$
```

The worst case[4] occurs when we merge half the clusters on the first iteration of the while loop, and a quarter of the points on the second iteration etc. Which means that in the worst case, we perform the while loop $O(logN)$ times. The VPTSearch is an $O(logN)$ operation. The pair of for loops are

[3]Vantage Point trees allow distance functions that are within a constant of a distance metric. The code in the supplemental material shows this enhancement.

[4]The points would need to be carefully crafted to result in this behaviour.

Algorithm 2 $HAC\_T\_opt(D, CDist)$

```
Step 1: Preprocess data
heap = NULL
**for** each item $A \in D$ **do**
    $VPTSearch(rootVPT, A, [A], best)$
    heap.Insert $(A, best.Point, best.Dist)$
**end for**

Step 2: Cluster data
**for** each item $A \in D$ **do**
    Put $A$ in its own cluster
**end for**
**while** $heap.nelem() > 0$ **do**
    $(A, B, dist) \leftarrow heap.deleteTop()$
    **if** $dist < CDist$ AND $cluster(A) <> cluster(B)$ **then**
        $Merge(cluster(A), cluster(B))$
    **end if**
**end while**

Step 3: Find clusters which need to be merged
$Clusters\_to\_examine \leftarrow All\_clusters$
**while** $len(Clusters\_to\_examine) > 0$ **do**
    $lmodified \leftarrow []$
    **for** each cluster $C \in Clusters\_to\_examine$ **do**
        **for** each item $A \in C$ **do**
            $VPTSearch(rootVPT, A, listMembers(Cluster(C)), best)$
            **if** $best.Dist \leq CDist$ **then**
                $newCluster \leftarrow Merge(cluster(A), cluster(best.Point))$
                $lmodified.append(newCluster)$
            **end if**
        **end for**
    **end for**
    $Clusters\_to\_examine \leftarrow lmodified$
**end while**
```

resulting in an $O(NlogN)$ operation. So the worst case for Step 3 overall is $O(Nlog^2N)$ operations. Step 3 is dominating the time complexity, while identifying long "stringy" clusters which are less likely to occur with the high dimensional data.

1.2.2 Edge Cases

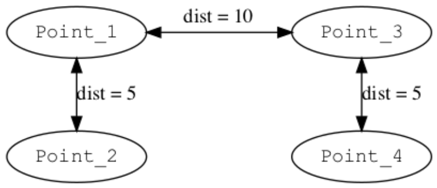

Figure 1: Edge Case for Step 2.

Step 2 will fail to merge some items which are near neighbors (with $dist \leq CDist$). This occurs when 2 pairs of points are closer to each other than to the other pair of points as shown in Figure 1. With a CDist=20 all 4 points in the figure should be merged. However, Step 2 will create 2 clusters:

- (Point_1, Point_2)
- (Point_3, Point_4)

1.2.3 HAC-T

To create a faster algorithm, we augment steps 1 and 2 to cover the important edge cases, and remove Step 3. To do this we modify:

- Instead of doing the VPTSearch at the beginning, we do this search just before we Merge clusters in step 2. We are exploiting the modification done to VPTSearch we introduced above where we can efficiently exclude points which are in the same cluster as the point under consideration.
- We identify clusters which have a nearby point belonging to another cluster. We define points (A, B) as being "nearby" when

  $$CDist < dist(A, B) \leq 2 \times CDist$$

  We record these in a tenative heap ($tent\_heap$). We note that 2 is a parameter which could be adjusted for various applications.
- After we have processed all points identifying points within our cluster distance ($CDist$), we process the $tent\_heap$. In the $TentativeMerge$ procedure, we examine all points in a cluster to determine if they are within $CDist$ of points belonging to other clusters. If we find a pair of points within distance $CDist$, then we Merge the clusters for those points.

Algorithm 3 $HAC\_T(D, CDist)$

```
Step 1: Initialise
$tent\_heap = NULL$
**for** each item $A \in D$ **do**
    Put $A$ in its own cluster
**end for**

Step 2: Cluster data
**for** each item $A \in D$ **do**
    $VPTSearch(rootVPT, A, cluster[A], best)$
    **if** $cluster(A) <> cluster(best.Point)$ **then**
        **if** $d \leq CDist$ **then**
            $Merge(cluster(A), cluster(best.Point)$
        **else if** $d \leq 2 * CDist$ **then**
            $tent\_heap.Insert(A, best.Point, best.Dist)$
        **end if**
    **end if**
**end for**
**while** $tent\_heap.nelem() > 0$ **do**
    $(A, B, dist) \leftarrow tent\_heap.deleteTop()$
    **if** $cluster(A) <> cluster(B)$ **then**
        $TentativeMerge(cluster(A), cluster(B)$
    **end if**
**end while**
```

The HAC-T algorithm does not guarantee an optimal single link clustering. That is there may be distinct clusters (cA, cB) where

$$A \in cA \quad AND \quad B \in cB \quad AND \quad dist(A, B) \leq CDist$$

For the applications we have studied, this occurs rarely (more details in the results section).
We expect that the while loop calling TentativeMerge will be doing minimal work, which occurs if a reasonable distance function and $CDist$ parameter are used. If the main for loop (calling Merge) dominates, then the time complexity is $O(NlogN)$, which is suitable for clustering large datasets.

2 Implementation and Results

We implemented the pseudo code in this paper as hac-t.py at
https://github.com/trendmicro/tlsh/tree/master/tlshCluster . The code provided closely follows the

structure here in the paper. In addition, in that repository we provide scripts and Jupyter notebooks for processing and understanding the Malware Bazaar dataset.

2.1 Data

In these experiments, we report on analysis of 132,134 data points collected from Malware Bazaar [3][5]. This data has the TLSH digest which gives a 132 dimensional representation of the malware file [3]. Each datapoint represents a malware file, and a malware label is provided[6]. The labels are assigned by submitters, and the dataset has 240 distinct non-empty labels.

The 132 dimensional representation is based on random projections of the files bytestream. These features are independant of the labels [19]. The distance function derived from these features is available on both Github [9] and Pypi.org [21] under either the Apache or FreeBSD license.

2.2 Experiment 1: Cluster Quality

We consider a range of CDist values (0-100)[7] and do repeated experiments on 10K of the dataset to determine the consistency of the labels within clusters. A cluster is defined as being cluster-pure if it has a single label and as cluster-impure if it contains 2 or more distinct labels [8]. In addition, we measure the percentage of the dataset being clustered and that we are getting a reasonable number of clusters. Figure 2 shows this data as we vary the $CDist$ parameter.

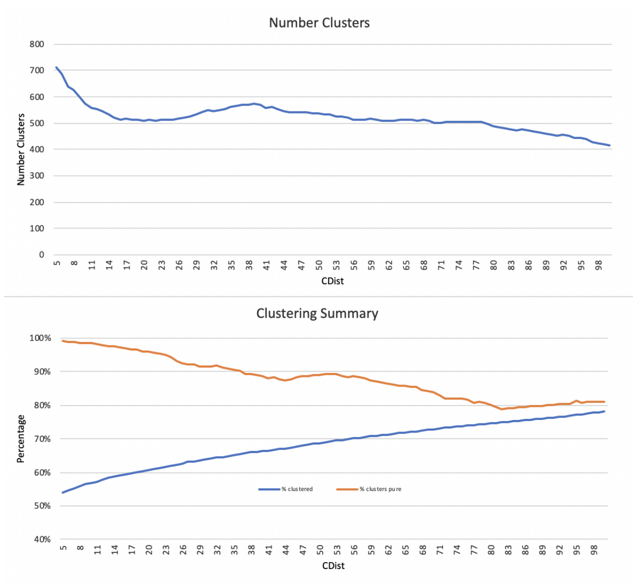

Figure 2: Cluster characteristics versus CDist parameter

The recommended cluster radius for the distance function provided is in the range (10-50) and possibly going up to 100. The plot shows cluster purity in the range 89% - 98%. Given that the data set has 240 distinct labels, then this is good-excellent cluster quality. Ad-hoc inspection of the clusters backs up this interpretation.

For this application, we can set the $CDist$ parameter to allow different cluster densities and sizes:
very pure clusters ($purity \geq 95\%$) with a lower coverage ($coverage \leq 60\%$), to
less pure clusters ($purity \leq 79\%$) with better coverage ($coverage \geq 75\%$).

[5]The ToS for the data from Malware Bazaar are compatible with research uses. Refer to [4].

[6]Malware labels are generally considered as being somewhat unreliable [20, 1].

[7]The range 0 to 100 in recommended in the definition of the LSH [19].

[8]For some clusters, we find multiple labels that relate to the same malware family. For the purposes of this research, we leave these clusters as cluster-impure.

2.3 Experiment 2: Timing

We ran the Python code[9] for datasizes varying from 5,000 to 60,000 and measured the total time taken to cluster the dataset for both HAC-T and HAC-T-opt with the CDist parameter set to 30. The time measured also includes the time taken to build the VPT. Figure 3 shows the time taken versus datasize graph. For HAC-T, we see timing consistent with the time complexity $O(NlogN)$. For HAC-T-opt we see better performance than the worst case analysis, and closer to the overall shape of an $O(NlogN)$. It appears that Step 3 of HAC-T-opt is using approximately half the time used.

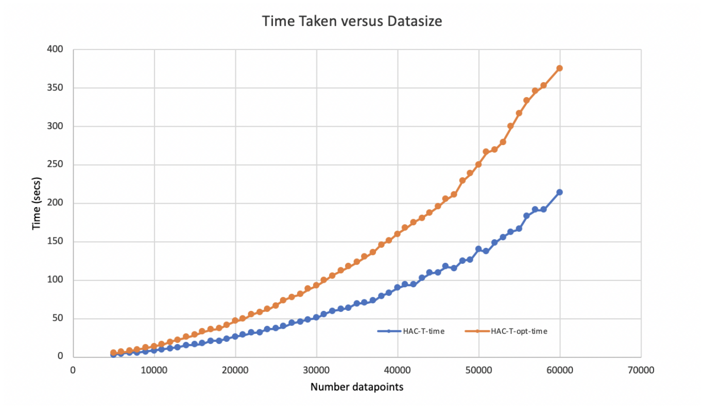

Figure 3: Time taken versus datasize graph

This experiment generated a total of 69490 clusters and the HAC-T approach generated a total of 48 clusters which were not identical to the HAC-T-opt. Each of those cases was a situation of two tight clusters with a single link of greater length consistent with the situation discussed in the Edge Case section.

2.4 Experiment 3: Cluster Density

We examined the clustering of the 60,000 sized dataseta (again with the CDist parameter set to 30) and plotted the cluster radius versus number of items in the cluster, as shown in Figure 4.

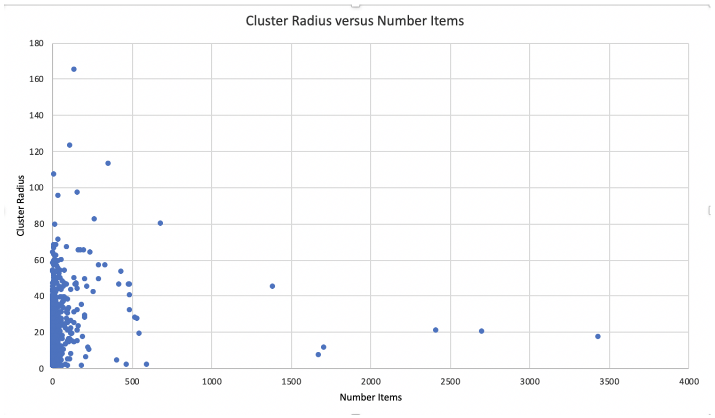

Figure 4: Cluster radius versus number of items plot

We see the majority of clusters have a cluster radius ≤ 60 (which is twice the single linkage allowed), and $nitems \leq 500$. Some clusters of interest are:

1. A cluster with 3431 items and a radius of 17 with nearly all having the label of NO_SIG.
2. A cluster with 2700 items and a radius of 20. 2696/2700 of them were labelled Quakbot.

[9]We have used Python code for comprehensibility. Table IV of [18] reports a C++ version using a VPT approach as clustering 1 million samples in 478 seconds on a standard AWS instances.

3. A dense cluster with 598 items and a radius of 2. This was uniformly labelled Trickbot.
4. The "stringiest" cluster contained 137 items, had a radius of 165 and 7 distinct labels.

This plot highlights the problems with selecting a single parameter to cover a range of cluster densities.

2.4.1 Understanding and Using the Clustering

The clustering created by these programs has been depicted as dendrograms at https://github.com/trendmicro/tlsh/blob/master/tlshCluster/malbaz.ipynb .

In Figure 5 we show various clusters of Ficker Stealer malware in the form of a dendrogram. The naming of the clusters consists of 3 elements

- The name of the majority malware family in the cluster (as identified by a variety of threat experts associated with Malware Bazaar);
- The first seen data for the cluster (the date when the first sample in the cluster was submitted); and
- The number of samples in the cluster;

The dendrogram gives us a compelling depiction of the data from a number of perspectives:

- There is consistency in the closest clusters being related (this is a known feature of malware. Malware authors are constantly mutating their malware to avoid detection.); and
- We find close clusters are associated in time which reflects that the mutation of malware grows as time goes on.

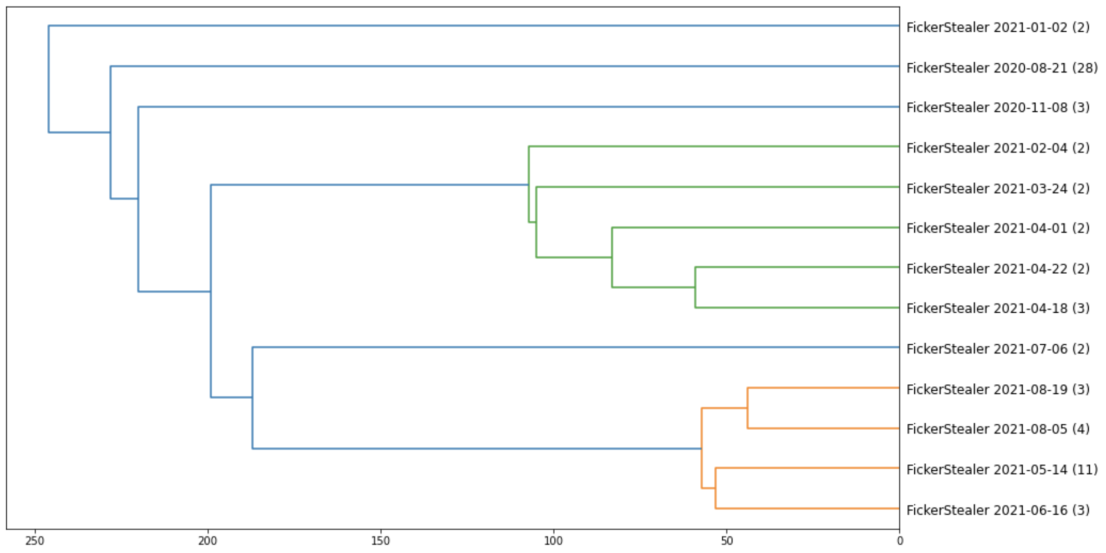

Figure 5: A dendrogram of the FickerStealer malware family

In Figure 6 we show various clusters centered on a specific TLSH value: T14893F844FD459B2FC3D372F6E75C028D763A1FE8A7E630269934BEA023F56D12526911. We see that the clusters near this value include both the Gafgyt and Mirai malware families. We see that there is a large branch of Gafgyt malware clusters at the top of the diagram and a large branch of Mirai malware clusters at the bottom of the diagram. This makes perfect sense. It was reported in April 2021, that Gafgyt had started re-using Mirai code https://securityaffairs.co/wordpress/116882/cyber-crime/gafgyt-re-uses-mirai-code.html .

We have depiected 2 dendrograms here, but we not that this feature of the clusters showing useful relationships occurs accross many malware families which are in the Malware Bazaar dataset. The

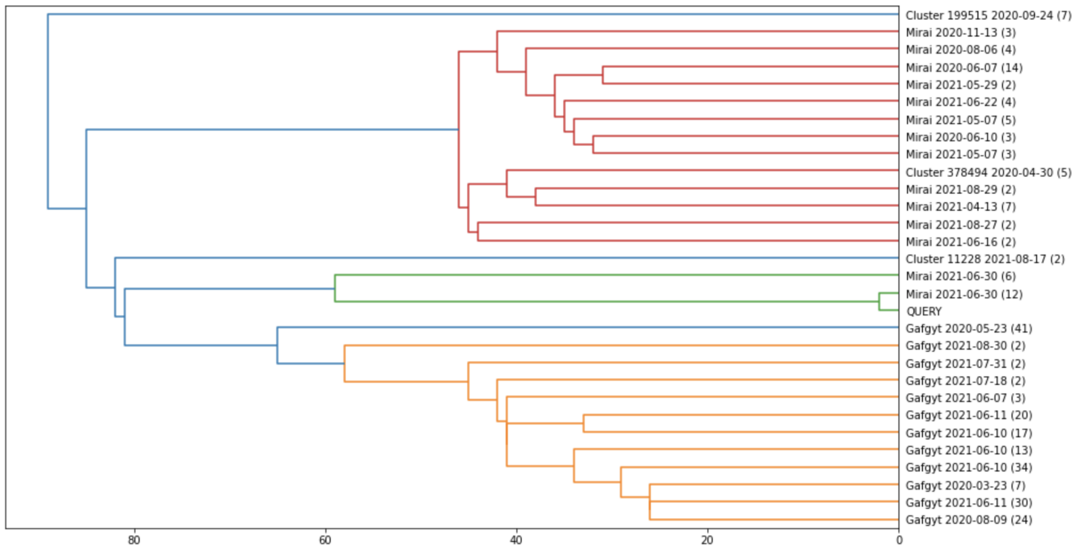

Figure 6: A dendrogram of the Gafgyt and Mirai malware families

relationships found are consistent with computer security background knowledge and offer useful guides to understanding how cybercriminals are mutating their malware.

3 Conclusions

We developed algorithmically fast clustering approaches for high dimensional data with an associated distance function. We used a distance function based on a fine-grain LSH where the clusters are larger than the LSH-bucket size (represented by distances > 1). We develop HAC-T-opt an optimal single-linkage clustering algorithm with time complexity $O(Nlog^2N)$. We adapted the HAC-T algorithm that reduce the time complexity to $O(NlogN)$, but we lost the guarantee of optimal clustering.

We explored the use of the TLSH distance measure in combination with the HAC-T algorithm on Malware Bazaar (a public data corpus of high dimensional malware data). We verified that this resulted in clustering which is consistent with expert labelling and provided near optimal clustering with time complexity $O(NlogN)$. In addition, we found that approaches described afford significant insights into the data.

References

[1] Johari Abdullah and Navein Chanderan. Hierarchical density-based clustering of malware behaviour. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(2-10):159–164, 2017.

[2] Muqeet Ali, Josiah Hagen, and Jonathan Oliver. Scalable malware clustering using multi-stage tree parallelization. In *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6. IEEE, 2020.

[3] Malware Bazaar. Malware Bazaar. `https://bazaar.abuse.ch/`, 2021. [Online; accessed 10-May-2021].

[4] Malware Bazaar. Malware Bazaar. Terms of service. `https://bazaar.abuse.ch/faq/`, 2021. [Online; accessed 10-May-2021].

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[8] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[9] Github. Tlsh software. `https://github.com/trendmicro/tlsh/`.

[10] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[11] Yu Hua, Bin Xiao, Dan Feng, and Bo Yu. Bounded lsh for similarity search in peer-to-peer file systems. In *2008 37th International Conference on Parallel Processing*, pages 644–651. IEEE, 2008.

[12] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[13] Hisashi Koga, Tetsuo Ishibashi, and Toshinori Watanabe. Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing. *Knowledge and Information Systems*, 12(1):25–53, 2007.

[14] Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *European conference on computer vision*, pages 364–378. Springer, 2008.

[15] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Intelligent probing for locality sensitive hashing: Multi-probe lsh and beyond. 2017.

[16] Miguel Martín-Pérez, Ricardo J Rodríguez, and Frank Breitinger. Bringing order to approximate matching: Classification and attacks on similarity digest algorithms. *Forensic Science International: Digital Investigation*, 36:301120, 2021.

[17] Frank Nielsen, Paolo Piro, and Michel Barlaud. Bregman vantage point trees for efficient nearest neighbor queries. In *2009 IEEE International Conference on Multimedia and Expo*, pages 878–881. IEEE, 2009.

[18] 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. IEEE, 2020.

[19] Jonathan Oliver, Chun Cheng, and Yanggui Chen. Tlsh–a locality sensitive hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13. IEEE, 2013. `https://documents.trendmicro.com/assets/wp/wp-locality-sensitive-hash.pdf`.

[20] Radu-Stefan Pirscoveanu, Matija Stevanovic, and Jens Myrup Pedersen. Clustering analysis of malware behavior using self organizing map. In *2016 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (CyberSA)*, pages 1–6. IEEE, 2016.

[21] PyPi. Tlsh python library. `https://pypi.org/project/py-tlsh/`.

[22] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[23] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

[24] Eliezer Silva, Thiago Teixeira, George Teodoro, and Eduardo Valle. Large-scale distributed locality-sensitive hashing for general metric data. In *International Conference on Similarity Search and Applications*, pages 82–93. Springer, 2014.

[25] Jeffrey Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4):175–179, 1991.

[26] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, 1993.