COUGAR: Clustering Of Unknown malware using Genetic Algorithm Routines

Zachary Wilkins
Dalhousie University
Computer Science
Halifax, Nova Scotia, Canada
zachary.wilkins@dal.ca

Nur Zincir-Heywood
Dalhousie University
Computer Science
Halifax, Nova Scotia, Canada
zincir@cs.dal.ca

ABSTRACT

Through malware, cyber criminals can leverage our computing resources to disrupt our work, steal our information, and even hold it hostage. Security professionals seek to classify these malicious software so as to prevent their distribution and execution, but the sheer volume of malware complicates these efforts. In response, machine learning algorithms are actively employed to alleviate the workload. One such approach is evolutionary computation, where solutions are bred, rather than built. In this paper, we design, develop and evaluate a system, COUGAR, to reduce high-dimensional malware behavioural data, and optimize clustering behaviour using a multi-objective genetic algorithm. Evaluations demonstrate that each of our chosen clustering algorithms can successfully highlight groups of malware. We also present an example real-world scenario, based on the testing data, to demonstrate practical applications.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; • **Computing methodologies → Genetic algorithms**; **Cluster analysis**;

KEYWORDS

Cyber security, Machine learning, Malware, Clustering, Cyber attack, Evolution, Multi-objective optimization

ACM Reference Format:
Zachary Wilkins and Nur Zincir-Heywood. 2020. COUGAR: Clustering Of Unknown malware using Genetic Algorithm Routines. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3390151

1 INTRODUCTION

As the world has become more dependent on computers, so too has their value risen in the eyes of cyber criminals. Increasingly, malicious actors have sought to employ widely available malware to inflict damage or extract financial rewards. This scales from consumers and small businesses, which are especially vulnerable to ransomware attacks [3], to large technology companies, such as DynDNS, which suffered a major denial-of-service (DoS) attack that crippled large parts of the global internet [7].

Furthermore, malware mutation engines have allowed for an unending torrent of binaries that are difficult to detect with traditional, signature-based anti-virus software [10]. Polymorphic malware, software that can change their abilities, are also a growing problem [4]. It has been estimated that in 2014 over 1,000,000 new malware samples were introduced every day, a 26% increase from the previous year [11]. This growth makes scaling manual malware analysis impossible.

As a result, it is necessary to fight computers with computers. One approach that has proven successful is clustering [11] [10] [4]. While there are innumerable ways to disguise malicious files, their attack and data exfiltration techniques are finite [11]. By clustering based on malicious behaviours, patterns, and severity, many malware may be reduced to a single grouping. In doing so, newly observed malware can be compared to their already classified peers, allowing for manual analysis to be triaged to the most severe samples.

In this paper, we design and develop a new system called COUGAR (Clustering of Unknown malware using Genetic Algorithm Routines). COUGAR reduces high-dimensional behavioural data from malware to two-dimensions, and optimizes clustering behaviour using a multi-objective genetic algorithm. After determining optimal configurations, a testing dataset is labelled for evaluation. We also explore a hypothetical situation applying the system to a real-world scenario. The results indicate that all of our clustering algorithms succeed to varying degrees, while the predicted labels leave room for growth. The malware analyst scenario demonstrates that such a system does indeed have practical, real-world applications.

The rest of this paper is organized as follows. We begin by surveying related research in Section 2. Following this, we discuss the dataset, feature processing techniques, and evolutionary multi-objective optimization procedures used in Section 3. We then present and discuss the results of the experiments in Section 4. Section 5 concludes our observations and suggests directions for future work.

2 RELATED WORK

Malware clustering has been explored by several works in the literature. Beginning with a survey of the field, Faridi et al. conduct an extensive study to determine optimal clustering algorithms, parameters, and features for malware clustering [5]. They run 5,673 malware from an industry partner in a Cuckoo sandbox, and collect network communications and system calls. They construct

GECCO '20, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3390151

ground truth reference clusters using triggered alerts from Suricata, a network threat detection engine. Alerts that belong to one family uniquely are taken as the label for that sample. This process yields 94 clusters. After extracting a variety of features from the collected Cuckoo information, they group the features into distinct categories. They reduce these features to the most important strings, and vectorize the corpus using the Term Frequency – Inverse Document Frequency (TF-IDF) technique. They employ a number of distance metrics as well as constructing dissimilarity matrices directly from the vectorization. The authors then engage a variety of clustering algorithms, and present an assortment of clustering metrics. They indicate that network features were the most successful, while including all features lead to the worst performance. As far as clustering algorithms were concerned, DBSCAN with NCD, spectral with cosine similarity, and hierarchical with Braycurtis and average linkage provided the best overall performance.

Employing Cuckoo sandboxes for extracting behavioural information is a popular technique, also used in [4], [11], and [19]. Each of these works use the extracted API call information as input for feature generation, but their output differs given the different approaches they use. In [4], Duarte-Garcia et al. construct a system to characterize malware behaviour, with the intent of assigning labels to unseen samples. To filter out noise, API calls common to at least 90% of malware are removed. They employ TF-IDF, as in [5], conceptualizing each malware as a document, and each call as a term in that document. k-means is employed, varying the number of clusters and evaluating cluster quality using silhouette coefficient. They then engage a number of classifiers. Their results show that 60 clusters was the most successful configuration, and gradient-boosting decision trees were the most successful classifier.

Lee et al. instead concoct an "intelligent analysis" system to guide malware analysts in their triage efforts against malware mutants [11]. API calls from Cuckoo are used to create bigrams representing call sequences. These bigrams are then compared using cosine similarity to facilitate grouping. Mutants that are above a given threshold t are joined together in a graph. This process results in 210 groups of malware. The top 10 groups, however, contain 43% of the samples, which can be automatically categorized as mutants.

Self-organizing maps (SOMs) are the tool of choice by Pirscoveanu et al. to cluster malware behaviours [19]. From the aforementioned API calls, they extract features representing successful/unsuccessful calls, and the return codes from failed calls. They then apply principal component analysis (PCA), a form of dimension reduction, to reduce the feature set. After using gap statistics to choose the number of clusters, they employ SOMs to project each sample onto a two-dimensional map, where the number of clusters is equal to the number of nodes in the map. As ground truth, they label each malware by collecting their anti-virus labels from VirusTotal, and remove noise by faceting to the top five Microsoft AV labels. Their SOMs achieve accuracies of approximately 80% when using a 2x2 map, and 70% with a 4x4 map.

Alternative clustering approaches include the use of distributed computation and fuzzy clustering, as described in [10] and [1]. In the former, Jang et al. create an automated, large scale malware triage system called BitShred to target malware mutants [10]. Employing feature hashing, they create a fingerprint for each malicious sample and calculate approximate Jaccard similarities. Clusters are created hierarchically based on a variable threshold. Co-clustering can then occur to discover correlated features in sub-groups of malware. Their results indicate that their approach is much faster than contemporary techniques, with similar accuracy. The latter focuses on mobile threats instead of the Windows platform, with Acampora et al. creating a system to discover Android malware phylogeny [1]. Using established Android malware datasets, they run 3,000 samples on a smartphone and collect system call traces, which are representative of the malicious behaviours. They transform these calls using a process modelling language called Declare, and create system call execution fingerprints (SEFs). From these SEFs, a dissimilarity matrix is calculated and passed to a fuzzy clustering algorithm, FANNY. This allows the researchers to determine the membership degrees of each malware family with respect to the determined clusters. They validate their results by establishing a reference clustering from previous literature, achieving precision and recall scores of 0.8 and 0.73, respectively.

Given the success of natural language approaches, we focus on similar techniques, especially taking inspiration from the intelligent triage intentions of [11] and the dimensionality reduction of [19]. Our novel contribution is specifically enabled by employing genetic algorithms for optimization of clustering algorithms.

An issue all of these works have in common is their approach to feature extraction. Namely, each require the original malicious binaries to extract features that are appropriate for other systems. This makes comparisons on the same dataset very difficult. As well, distributing malicious files is fraught with peril, as their dangerous nature can cause discomfort to system administrators, and the responsible use of these dangerous files is difficult to verify when distributed openly. A solution to this issue is the use of an open and reputable, 3rd party feature dataset.

To that end, and in the interest of replicability, we engage an open dataset and open-source[1] our code. We hope that this transparency encourages more direct comparison between works in this domain, as they are presently hindered by a lack of open-source tools and the use of closed-source datasets.

3 METHODOLOGY

In this section, we describe the dataset, processing techniques, and algorithms employed in these experiments.

3.1 Dataset

For this work, we chose to use the relatively novel Endgame Malware BEnchmark for Research (EMBER) dataset [2] from Endgame. EMBER was designed as a "benchmark dataset for researchers" [20], and includes features extracted using LIEF [22] from over 2,000,000 Windows Portable Executables (PEs) in 2017 and 2018.

The features in EMBER are numerous. For samples from 2018 alone, the uncompressed JSON files are in excess of 9GB. These features include general file characteristics, as well as file header information, imported and exported functions, byte and byte-entropy histograms, and string statistics. We utilize the AVClass feature, obtained from the eponymic tool[2] as our label (ground truth). These labels are decided in an automatic process idempotent to the platform,

[1] https://github.com/znwilkins/cougar
[2] https://github.com/malicialab/avclass

or anti-virus vendor, and can be computed without the malicious executable by hooking into online APIs.

It has been shown [13] [23] that import statements are an effective tool to extract malicious behaviours, as it is difficult to obfuscate 3rd party imports without breaking compatibility with the library. To that end, import terms and their corresponding library were extracted from EMBER for each malicious sample. In the interest of speed, the container format was also modified during extraction. Representing imports as a table allows for compression advantages in the form of repeated values. For this task, Apache Parquet[3] was chosen. Parquet is a column-oriented data format optimized for the Hadoop ecosystem, where in-memory data access is paramount, an attribute shared with COUGAR. The MD5, SHA256, assigned AV-Class, and total number of imports were also included in a separate table for reference.

3.2 Feature Processing

The transformation from EMBER to coordinates is described in this subsection. A high-level visual representation can be observed in Figure 1.

3.2.1 Vectorization. Imports from each sample are converted to lowercase, joined with a comma to their parent library, and added to the corpus. A Bag-of-Words model is employed to vectorize the dataset, using the `CountVectorizer` from Scikit-Learn [18]. In the resulting matrix, each row represents a single malicious sample, and each column counts the number of occurrences for each library/function in the corpus. Since EMBER only contains a list of function imports, rather than frequency of function calls, the resulting sample representation is a bit vector, where the presence of a term is indicated by a set bit. This vectorization is then held in memory for further computation.

3.2.2 Dimension Reduction. While clustering on higher dimensions is possible, the nature of a production malware triage system is continuous data ingestion. To this end, updated cluster models can be produced significantly faster on low-dimensional data.

The algorithm we employ for this task is the Uniform Manifold Approximation and Projection (UMAP) technique [17]. This technique works by modelling the data with a fuzzy topological structure, and is notable for attempting to preserve both local and global distances between data points.

UMAP was chosen for a number of reasons. UMAP demonstrates certain qualities that make it better suited as a cluster preprocessing step than other dimension reduction techniques like t-SNE. Namely, it more effectively maintains the "global structure" of the data which leads to more "meaningful separation between connected components of the manifold on which the data lies" [14]. In turn, the resulting clusters in the embedding are more relevant. As well, it has been demonstrated to be much faster than similar techniques such as PCA and t-SNE in benchmarks [15], which is an important future consideration to scale the system to handle the large influx of malware every day. Finally, UMAP supports the transformation of new data into existing models [16]. While this feature is not unique to UMAP, it is essential for our use case in predicting the malware family of unseen samples based upon some training dataset. A pre-release version[4] of UMAP is used for its expanded support of sparse matrices. While UMAP can be parameterized to reduce to any number of dimensions, we elected to reduce to a two-dimensional embedding for the sake of simplicity, and ease of visualization. We set the number of neighbours for UMAP to 1% of the training dataset (20), and all other parameters were set to default. Neighbourhood size is an important consideration, as it determines the degree to which the algorithm will focus on local or global structure.

3.3 Evolutionary Multi-Objective Cluster Optimization

To determine the most effective way that these data points can be grouped together, it was necessary to utilize evolutionary multi-objective optimization (EMO). The result of this process is a sequence of parameters for the chosen clustering algorithms.

3.3.1 Clustering Algorithms. We employ three clustering algorithms in our experiments. We selected DBSCAN, a density-based clustering technique, as it supports non-convex cluster shapes, as well as *not* clustering data that it perceives to be noise. The former consideration is especially important when working with UMAP, as it is not guaranteed to produce "clean, spherical clusters" [14]. The latter is also a major benefit in this research problem, as unrelated samples with close proximity to legitimate clusters can be discarded, rather than forcing their inclusion.

A closely related technique, OPTICS, is also engaged for comparison, as it is considered to be better suited[5] for large datasets than Scikit-Learn's DBSCAN implementation. In addition, neither DBSCAN nor OPTICS requires parameterization with the number of clusters in the dataset. This mitigates a 'chicken and egg' scenario, where the number of clusters is required to cluster the data, but a reasonable estimate of cluster count is difficult without first clustering the data.

The eminently popular k-means is also considered. While we did not anticipate great performance out of k-means, given its demonstrated weakness on data exhibiting characteristics similar to UMAP output [21], the longevity of the technique still gives it comparative value as a baseline clustering algorithm.

For each algorithm, two or three parameters were selected for optimization. DBSCAN uses `eps` and `min_samples`, while OPTICS uses `max_eps` and `min_samples`. k-means uses `n_clusters`, `n_init`, and `max_iter`. These were chosen in concert with the documentation for Scikit-Learn, as they identify the most consequential parameters. Each algorithm is initialized using reasonable parameters in the vicinity of their defaults in the Scikit-Learn implementation. The notable exception to this is `min_samples` in DBSCAN and OPTICS, which is set higher (40), to avoid forming unhelpfully small clusters.

3.3.2 Optimization Objectives. There are three objectives guiding parameter tuning. Intuitively, we want as many good clusters as possible, containing the right points in the right clusters, and we want those clusters to be large. More formally, we seek to:

[3] https://parquet.apache.org

[4] https://github.com/lmcinnes/umap/tree/d214e5dbaa30f63a0bf7608d9de96cfa94de21e1

[5] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html

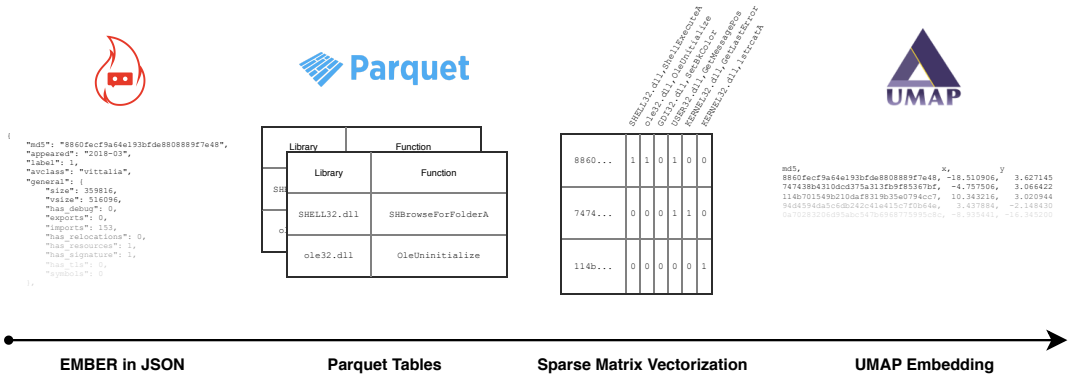

Figure 1: A visual depiction of the transformation from EMBER to embedding.

(1) Maximize the count of high-quality clusters. The quality of a cluster is determined by computing the cosine similarities over all elements in the cluster, using the vectorization described in 3.2.1. Given malicious bit vectors **A** and **B** of length n, we calculate cosine similarity as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=0}^{n} A_i B_i}{\sqrt{\sum_{i=0}^{n} A_i^2}\sqrt{\sum_{i=0}^{n} B_i^2}}$$

These calculations are summed and scaled to the range $[0.0, 1.0]$. A high-quality cluster is achieved if the scaled summed similarity (SSS) is at least 0.8.

(2) Minimize the sum of the sum squared error (SSE) for each cluster. The SSE is calculated as follows:

$$SSE = (1 - SSS)^2$$

(3) Maximize the median cluster size. Median was chosen over average to avoid clustering parameters that would create a small number of unreasonably large clusters, thereby skewing the metric and incorrectly indicating success.

3.3.3 Optimization Algorithm. We employ the Non-dominated Sorting Genetic Algorithm III (NSGA-III) to optimize the aforementioned objectives. NSGA-III is an extension of NSGA-II to many-objective problems, employing a reference set of points to achieve a uniform Pareto front of non-dominated solutions [9]. The implementation of NSGA-III was handled by the Distributed Evolutionary Algorithms in Python (DEAP) library [6], which notably supports out-of-the-box parallelization through Scalable COncurrent Operations in Python (SCOOP) [8].

To parameterize the algorithm, we must consider the partitions (p) of the reference plane, and the number of objectives ($M = 3$). [9] gives the total number of reference points as:

$$H = \binom{M + p - 1}{p}$$

The same work also suggests $p = 12$ for the number of partitions in the reference plane, but this leads to a reference points size of $H = 91$. The resulting population size, which is calculated to be the smallest multiple of 4 greater than H, per [9], is 92. This dramatically slows the evolutionary process due to the large size and slow fitness evaluation, courtesy of cosine similarity's $O(n^2)$ time complexity. Choosing $P = 4$ is much more reasonable, leading to $H = 15$ and a final population size of 16. The algorithm is run for 100 generations, with a crossover probability of 1.0, an individual mutation probability of 1.0, and a gene mutation probability of 0.25.

Individuals are represented with two or three floating points numbers, according to the parameters of the given clustering algorithm. To account for the variable scales upon which the parameters can be specified, we scale the floating point gene, and optionally convert it to an integer, before passing it to the clustering algorithm. For example, to evaluate the default DBSCAN individual <0.005, 0.04>, the first gene would be multiplied by 100, and the second would be multiplied by 1000 and converted to an integer. This would then be interpreted as `eps= 0.5` and `min_samples= 40`.

In the following evaluations, we train each clustering algorithm with a training dataset of samples from EMBER to determine optimal parameters. Following this, we select an algorithm for further scrutiny, with a testing dataset of additional samples. Through this process, we work to ascertain the potential of labelling unseen data with COUGAR.

4 EVALUATIONS

To obtain optimal parameters for the clustering algorithms, they were each trained on 2,000 samples from EMBER. Accounting for the stochasticity of genetic algorithms, this process was repeated 10 times. After completing these runs, statistical analysis was performed via analysis of variance (ANOVA) tests, to determine significant differences from the mean for each objective. For those that reject the null hypothesis, additional post hoc testing was conducted. This yielded a leading algorithm, which was then subject to additional training, before introduction of a testing dataset of 1,000 samples for labelling. From this labelled data, final metrics were calculated.

4.1 Training Results

Each individual produced over all training runs was ranked according to their objective scores. The ordering was determined by the count of quality clusters, deferring to summed SSE, and then median size, in case of ties. In addition, a number of metrics were calculated to determine how well labelled these clusters are. The

labelling process proceeded as follows. For each output cluster, all sample record information was gathered, including the original AVClass. Then, each sample in the cluster had a predicted label assigned according to the majority vote of original AVClass labels in the cluster.

With these predicted and actual AVClass labels, weighted F-Score, homogeneity, completeness, and V-Measure were calculated. We now define these metrics.

To understand F-Score, we must first define precision and recall. Precision is a measure of the relevancy of the results, while recall is a measure of the quantity of relevant results returned. Given counts of true positives (T_p), false positives (F_p), true negatives (T_n), and false negatives (F_n), precision and recall are:

$$P = \frac{T_p}{T_p + F_p} \qquad R = \frac{T_p}{T_p + F_n}$$

F-Score is then the harmonic mean of precision and recall:

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

Since we have multiple classes, our reported F-Scores are the averaged F-Scores for each class, weighted by support. Given a set of F-Scores for all n classes (F_C), from N data samples, and a set of support values (S) for each class, we have:

$$F_w = \frac{1}{N} \sum_{i=0}^{n} F_{C\,i} \cdot S_i$$

Weighting the F-Score by support for each label was necessary, as not every potential label in EMBER is represented in these training samples. Otherwise, this serves to falsely distort the F-Score, as achieving many F-Scores of 0.0 on labels without any samples is not helpful when assessing performance.

A clustering is considered perfectly homogeneous if each of its clusters only contain data points from a single class. It is considered perfectly complete if all data points from a given class are contained in the same cluster. To state the mathematical definitions, given a set of classes (C), and a set of clusters (K), homogeneity is:

$$h = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases}$$

and completeness is:

$$c = \begin{cases} 1 & \text{if } H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{else} \end{cases}$$

where H is entropy. V-Measure is then the harmonic mean of homogeneity and completeness, defined as:

$$v = 2 \cdot \frac{h \cdot c}{h + c}$$

The top three clusterings for each algorithm, with these statistics, can be seen in Table 1. One characteristic associated with all three algorithms is a quick convergence to a local optimum, followed by regular scrubbing until the evolutionary process completes. This can be observed in Figure 2, where the green circle represents the start, and the red star represents the end.

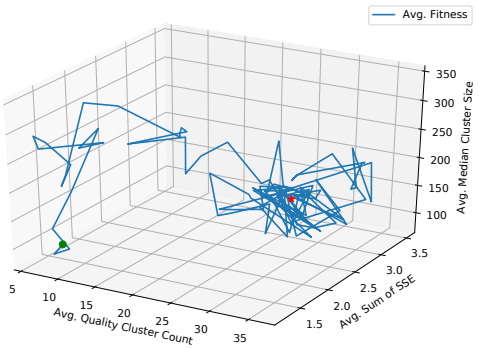

Figure 2: Avg. objective performance over one DBSCAN run.

4.1.1 DBSCAN. Looking first at trendlines for each objective in Figures 3, 4, and 5, we can see that DBSCAN is relatively flat and averaging 30 quality clusters by the conclusion of each evolutionary process. Its summed error is much the same, sitting just above 2.0. Median cluster size is the more variable objective, with lows of 200 and highs near 400, the trend somewhere in the middle around 325.

As far as its top performers are concerned, DBSCAN achieves over 60 quality clusters with each. More notably, it still attains low errors averaging around 4.06. The average median cluster size is 8.5 across the best performers, significantly lower than the average. This suggests that those high quality clusters are achieved by grouping smaller clusters of data points.

Looking at the metrics for the labelled data, we see consistent weighted F-Scores around 0.673. Homogeneity is higher, averaging at 0.735, but completeness is higher still, at 0.866. This seems to indicate that, while having some mislabeled data, the predicted groupings contain most of the instances for their given label. This naturally leads to a V-Measure in the middle, averaging 0.795.

4.1.2 OPTICS. In the trendlines, OPTICS achieves a marginally higher quality cluster count than DBSCAN, sitting at approximately 34. The variability of this objective is noticeable, however, with lows of 25 and highs approaching 40. In fact, this variability extends to the other objectives. The summed error ranges from below 3.0 to nearly 5.0, with the trend at just under 4.0. The spread on median cluster size is not as dramatic as DBSCAN. It ranges from above 200 to below 100, with the trend around 140.

Top performing clusterings are a mixed bag. The quality cluster counts are excellent, at 83 and 84, 20 clusters more than DBSCAN. Unfortunately, the summed SSE for those clusterings are nearly triple their DBSCAN peers, averaging 11.68. Median size is much the same as DBSCAN, if slightly lower, with an average of 7.6. All-in-all, these clusterings manage to find more high quality clusters, with the consequence of higher noise.

The metrics in Table 1 are lower, across the board, relative to DBSCAN. The previously mentioned noise drags the average weighted F-Score down to 0.547. Homogeneity and completeness are more successful, though, averaging 0.674 and 0.789, respectively. Their average V-Measure comes out to 0.727.

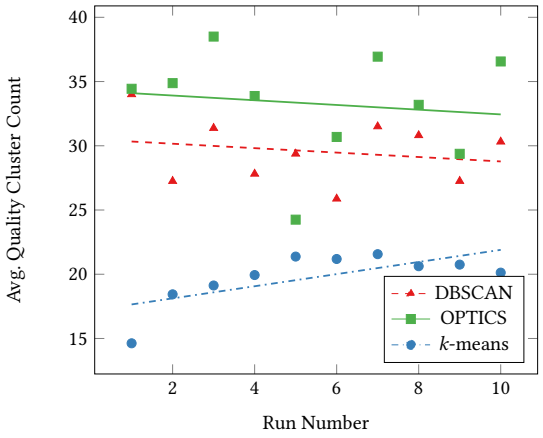

Figure 3: Avg. count of quality clusters over 10 training runs.

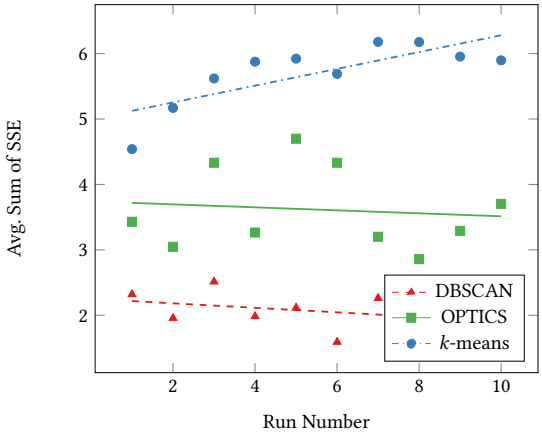

Figure 4: Avg. sum of SSE over 10 training runs.

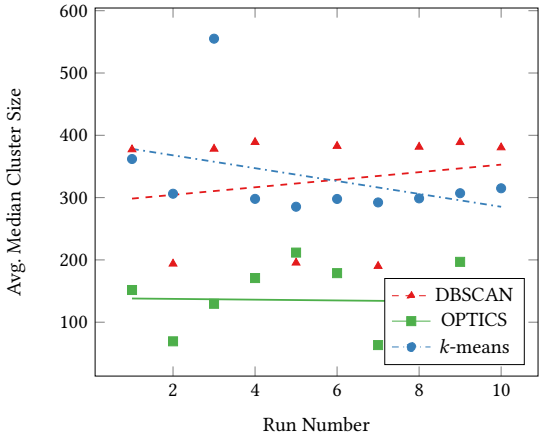

Figure 5: Avg. median cluster size over 10 training runs.

4.1.3 k-means. A final look at the trendlines show that *k*-means actually has the most consistent quality cluster count, where 9 of the 10 runs are very close to the trend at or around 20. This stability is lacklustre, however, as it is also the lowest average cluster count of the three algorithms. The average median cluster size is actually competitive with DBSCAN, at an average of 331, and fairly stable across 8 of the 10 runs. This could be an effect of every *k*-means parameter needing to be truncated to an integer, so minor mutations of the floating point genes do not result in dramatic changes to the clustering performance.

Given that *k*-means cannot label data as noise, and must include every point in some cluster, its best clusterings are impressively close to DBSCAN on the quality cluster front, with an average of 50. In fact, the gap between DBSCAN and *k*-means is smaller than the gap between DBSCAN and OPTICS, at approximately 13 clusters difference. Naturally, from the necessarily clustered data, the median cluster size is also the largest of the three algorithms, at 15.3. Unfortunately, the summed SSE is again the highest of all three algorithms, at 13.31 on average.

With such error, it comes as no surprise that the metrics on predicted labels are the worst of all three algorithms. The weighted F-Score is just under OPTICS, averaging at 0.525. Homogeneity is not much better, at 0.560. Completeness is more successful, though, just under OPTICS at 0.771. Their harmonic mean then averages to 0.668.

4.2 ANOVA

To determine which algorithm should be subject to further testing, it was necessary to apply analysis of variance (ANOVA) tests to the final average of each objective over all runs. For the quality cluster count, summed SSE, and median cluster size, the following P-Values were calculated: 5.83e-10, 5.52e-15, and 3.61e-06. With such small P-Values ($P \ll 0.05$), the null hypothesis is rejected for each objective, indicating statistically significant differences from the mean.

To determine the difference between the means for each algorithm in each objective, we ran Tukey's honestly significant difference (HSD) test on each objective. The results are in Table 2.

These tests confirm much of what we have discussed in 4.1. Namely, for the number of quality clusters, OPTICS narrowly edges out DBSCAN. For error, the reverse is true. For median cluster size, DBSCAN comes out on top, with a significantly larger median cluster size. In fact, the only comparison that does not show a statistically significant difference is the average median cluster size difference in Table 2 between DBSCAN and *k*-means which, as previously mentioned, is very similar.

Considering these realities, as well as the weighted F-Score, homogeneity, completeness, and V-Measure for each of the algorithms, it becomes clear that DBSCAN is the top performer. As a result, it was the natural choice for further evaluations.

4.3 Testing Data

We ran another 10 runs of COUGAR using DBSCAN on 2,000 EMBER samples, but this time preserving an additional 1,000 samples in the vectorization as testing data. All 160 individuals were scored, and one of the top 16 individuals was chosen, with similar objective scores to the DBSCAN clusterings in Table 1. This individual achieved scores of 60, 3.212, and 9.50 over the three

Algo.	Qual. Cnt.	Sum of SSE	Med. Size	Wgtd. F-Score	Homo.	Comp.	V-Meas.
DBSCAN 1	**65**	**3.88**	**8**	**0.668**	**0.725**	**0.862**	**0.788**
DBSCAN 2	62	3.87	9	0.678	0.748	0.872	0.805
DBSCAN 3	62	4.42	8.5	0.673	0.733	0.864	0.793
OPTICS 1	**84**	**10.83**	**8**	**0.549**	**0.675**	**0.792**	**0.729**
OPTICS 2	83	10.86	8	0.554	0.677	0.795	0.731
OPTICS 3	83	13.35	7	0.539	0.671	0.780	0.722
k-means 1	**52**	**13.62**	**15**	**0.520**	**0.593**	**0.774**	**0.671**
k-means 2	50	13.52	15	0.528	0.588	0.773	0.668
k-means 3	49	12.78	16	0.526	0.588	0.767	0.666

Table 1: Top three unique clusterings per algorithm, ranked by ascending objectives

Obj.	Algo. 1	Algo. 2	$\Delta\mu$	P-Adj.	Reject?
Qual. Cnt.	DBSCAN	*k*-means	-9.7812	0.001	True
Qual. Cnt.	DBSCAN	OPTICS	3.7125	0.0302	True
Qual. Cnt.	*k*-means	OPTICS	13.4938	0.001	True
Sum of SSE	DBSCAN	*k*-means	3.6401	0.001	True
Sum of SSE	DBSCAN	OPTICS	1.5532	0.001	True
Sum of SSE	*k*-means	OPTICS	-2.0869	0.001	True
Med. Size	DBSCAN	*k*-means	6.0812	0.9	False
Med. Size	DBSCAN	OPTICS	-190.4063	0.001	True
Med. Size	*k*-means	OPTICS	-196.4875	0.001	True

Table 2: Tukey's HSD test on each of the three objectives

objectives, with parameters of `eps=` 0.034971304580205594 and `min_samples=` 5.

At this point, we reloaded the embedding, and fit the new data into it. After re-clustering using the same settings as the chosen individual, we labelled the data as in 4.1, and calculated metrics on those predictions.

It is important to note that only approximately 30% of testing data gets clustered, as DBSCAN labels much of it as noise. As well, since clusters which only contain testing data do not get a label assigned, they must be discarded. Accordingly, only about 20% of the testing data is assigned a label.

With that said, the results can be observed in Table 3. We can see that the testing and training F-Scores are very similar, around 0.67, and this aligns with what we saw for DBSCAN in Table 1. Homogeneity and completeness are also nearly identical, with this training data ever so slightly higher than before, with an accompanying V-Measure.

When looking only at the testing data, we can see a slight drop in weighted F-Score, but the highest homogeneity, completeness, and V-measure across any of our reported values, at 0.821, 0.922, and 0.868, respectively. The implication is that, looking only at unseen data, the resulting groupings are fairly homogeneous, and highly complete, with the unseen data for their respective labels being grouped together.

Putting it all together, we see that the final count of quality clusters has improved, with median cluster size nearly unchanged, and summed SSE marginally increasing. There is a negligible decrease ($\leq 1\%$) across the label metrics, relative to training data only.

The takeaway here is that, while leaving room for improvement, the testing data predictions are as good on seen as on unseen data. Accordingly, a refinement in methodology should probably lead to a boost in accuracy in future work.

4.4 An Illustrated Example

While objective scores and metrics allow us to make reasonable comparisons between runs, it is also important to explore the resulting data from an analyst's perspective in the real world. Assuming a massive influx of samples on a particular day, it behooves malware analysts to concentrate on the most severe malware. Does COUGAR assist them in this process?

To find out, we drilled down on the assigned labels to the testing data.

Assume that one morning, our fictional analyst opens their daily COUGAR report and sees the following listed as the top three most numerous predicted malware family labels:

- installmonster: 61 instances
- high: 20 instances
- wannacry: 13 instances

The analyst recognizes 'installmonster' as adware with overall low risk[6], and therefore not their first priority. The second family, 'high', is an artifact of the string "high confidence" that is often assessed by anti-virus engines. These will take some time, as they will need to be individually examined. The third value, however, gives the analyst pause.

WannaCry is an aggressive ransomware family that caused much trouble in May 2017 [12]. This was especially true for hospitals in the United Kingdom that rely on legacy software designed for Windows XP [12], which has reached end-of-life. The analyst realizes that this must be their first priority.

[6] https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/ADW_INSTALLMONSTER/

Data	Qual. Cnt.	Sum of SSE	Med. Size	Wgtd. F-Score	Homo.	Comp.	V-Meas.
Training Only	60	**3.21**	**9.5**	**0.680**	0.734	0.870	0.796
Testing Only	–	–	–	0.671	**0.821**	**0.922**	**0.868**
Combined	**74**	5.42	9.0	0.673	0.727	0.870	0.792

Table 3: Objective scores and metrics for the final DBSCAN training / testing run

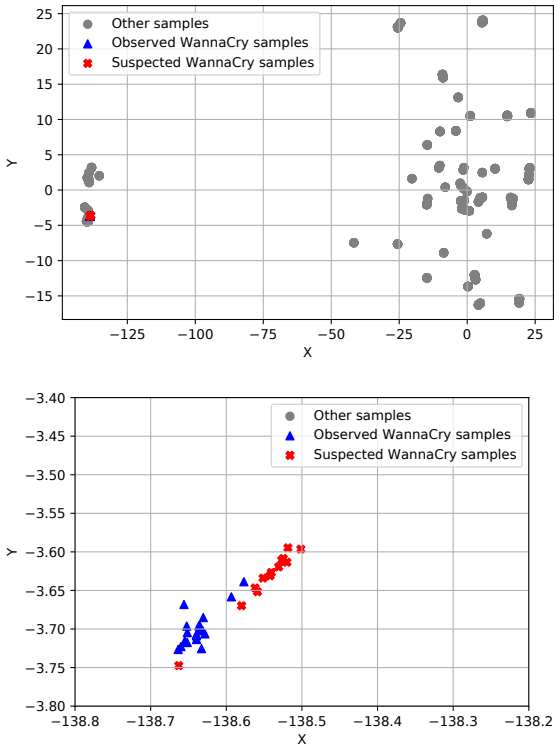

Figure 6: An analyst's visualization of WannaCry predictions from COUGAR.

They load the COUGAR UMAP embedding, and filter to only observed and suspected WannaCry samples. The data is concentrated in one location, and after adjusting bounds, the analyst can see that there are no other samples in close proximity to these new suspected samples. These graphs can be observed in Figure 6.

The analyst chooses one of these new samples and confirms that it is indeed a WannaCry variant. Since UMAP has operated on the vectorization of the malware behaviour, the analyst can be reasonably confident that these other samples are also WannaCry. They mark the samples as confirmed and carry on to their next task.

But how accurate was that assessment? As it turns out, very! According to the ground truth AVClass labels, 12 of those 13 predicted labels were indeed WannaCry. The mislabelled sample (MD5: `d0c40bf1a7ed5a417df63742acc1332c`) has actual AVClass 'silentinstall', and, upon further inspection at VirusTotal[7], is noted to behave like ransomware by McAfee Gateway Edition. To that end, a mislabelling to WannaCry is not a fundamentally wrong conclusion.

5 CONCLUSION AND FUTURE WORK

In this paper, our major contribution is the design, development and evaluation of a proof-of-concept system, COUGAR, to reduce high-dimensional behavioural data from malware to two-dimensions, and optimized clustering behaviour using a multi-objective genetic algorithm. After determining optimal configurations, a testing dataset was labelled for evaluation. We also explored a hypothetical situation applying the system to a realistic, real-world scenario. We believe that these experiments demonstrate the potential that exists in employing multi-objective genetic algorithms for tuning clustering algorithms in the domain of malware identification.

Our results indicate that each clustering algorithm is capable of producing at least satisfactory results, and highlight many high-quality clusters. OPTICS identifies the greatest number of clusters, while DBSCAN achieves the lowest summed sum of squared errors. k-means, while not spectacular, is not entirely uncompetitive on the quality cluster front, though lacks when it comes to error.

As far as predicted labels are concerned, there remains more work to be done, as weighted F-Scores never exceed 0.68. That said, the training and testing scores were extremely close, implying that when the labelling methodology improves, so too will the predicted labels.

Finally, in our real-world scenario, the analyst successfully identified the WannaCry samples with the help of COUGAR's predictions, which got 12 of 13 labels correct.

In future work, we would like to improve the label metrics by allowing for more granularity in label predictions. This could include fuzzy clustering logic, and clusters with more than one label, to mitigate scenarios where the majority vote is not actually a majority of the data. After this, the next logical step is to scale the system to larger sample sizes.

ACKNOWLEDGEMENTS

This research is partly supported by the Natural Science and Engineering Research Council of Canada (NSERC). This research is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/.

REFERENCES

[1] G. Acampora, M. L. Bernardi, M. Cimitile, G. Tortora, and A. Vitiello. 2018. A Fuzzy Clustering-based Approach to study Malware Phylogeny. In *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 1–8. https://doi.org/10.1109/FUZZ-IEEE.2018.8491625

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

[2] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018). arXiv:cs.CR/1804.04637

[3] Thomas Daigle. 2019. 'Definite uptick': Global wave of ransomware attacks hitting Canadian organizations. *CBC News* (Oct 2019). https://www.cbc.ca/news/technology/more-ransomware-canada-1.5317871

[4] H. L. Duarte-Garcia, A. Cortez-Marquez, G. Sanchez-Perez, H. Perez-Meana, K. Toscano-Medina, and A. Hernandez-Suarez. 2019. Automatic Malware Clustering using Word Embeddings and Unsupervised Learning. In *2019 7th International Workshop on Biometrics and Forensics (IWBF)*. 1–6. https://doi.org/10.1109/IWBF.2019.8739186

[5] Houtan Faridi, Srivathsan Srinivasagopalan, and Rakesh Verma. 2018. Performance Evaluation of Features and Clustering Algorithms for Malware. 13–22. https://doi.org/10.1109/ICDMW.2018.00010

[6] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (July 2012), 2171–2175.

[7] Sean Gallagher. 2016. DoS attack on major DNS provider brings Internet to morning crawl. *Ars Technica* (Oct. 2016). https://arstechnica.com/information-technology/2016/10/dos-attack-on-major-dns-provider-brings-internet-to-morning-crawl/

[8] Yannick Hold-Geoffroy, Olivier Gagnon, and Marc Parizeau. 2014. Once you SCOOP, no need to fork. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 60.

[9] H. Jain and K. Deb. 2014. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation* 18, 4 (Aug. 2014), 602–622. https://doi.org/10.1109/TEVC.2013.2281534

[10] Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. ACM, New York, NY, USA, 309–320. https://doi.org/10.1145/2046707.2046742

[11] Taejin Lee, Bomin Choi, Youngsang Shin, and Jin Kwak. 2018. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient. *The Journal of Supercomputing* 74, 8 (01 Aug. 2018), 3489–3503. https://doi.org/10.1007/s11227-015-1594-6

[12] Timothy B. Lee. 2017. The WannaCry ransomware attack was temporarily halted. But it's not over yet. (May 2017). https://www.vox.com/new-money/2017/5/15/15641196/wannacry-ransomware-windows-xp

[13] A. Martín, H. Menéndez, and D. Camacho. 2017. MOCDroid: Multi-objective evolutionary classifier for Android malware detection. *Soft Computing* 21, 24 (2017), 7405–7415.

[14] Leland McInnes. 2018. Frequently Asked Questions. (July 2018). https://umap-learn.readthedocs.io/en/latest/faq.html

[15] Leland McInnes. 2018. Performance Comparison of Dimension Reduction Implementations. (June 2018). https://umap-learn.readthedocs.io/en/latest/benchmarking.html

[16] Leland McInnes. 2018. Transforming New Data with UMAP. (July 2018). https://umap-learn.readthedocs.io/en/latest/transform.html

[17] L. McInnes, J. Healy, and J. Melville. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints* (Feb. 2018). arXiv:stat.ML/1802.03426

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[19] R. Pirscoveanu, M. Stevanovic, and J. M. Pedersen. 2016. Clustering analysis of malware behavior using Self Organizing Map. In *2016 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (CyberSA)*. 1–6. https://doi.org/10.1109/CyberSA.2016.7503289

[20] Phil Roth. 2019. Endgame Malware BEnchmark for Research Readme. (Sept. 2019). https://github.com/endgameinc/ember/blob/master/README.md

[21] John Sukup. 2018. When K-Means Clustering Fails: Alternatives for Segmenting Noisy Data. (Feb. 2018). https://blogs.oracle.com/datascience/when-k-means-clustering-fails%3a-alternatives-for-segmenting-noisy-data

[22] Romain Thomas. 2019. LIEF - Library to Instrument Executable Formats. (Nov. 2019). https://github.com/lief-project/LIEF

[23] Zachary Wilkins and Nur Zincir-Heywood. 2019. Darwinian Malware Detectors: A Comparison of Evolutionary Solutions to Android Malware. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*. ACM, New York, NY, USA, 1651–1658. https://doi.org/10.1145/3319619.3326818