A New Burrows Wheeler Transform Markov Distance

Edward Raff,[1,2,3] **Charles Nicholas,**[3] **Mark McLean**[1]
[1]Laboratory for Physical Sciences, [2]Booz Allen Hamilton, [3]University of Maryland, Baltimore County
raff_edward@bah.com, nicholas@umbc.edu, mclean@lps.umd.edu

Abstract

Prior work inspired by compression algorithms has described how the Burrows Wheeler Transform can be used to create a distance measure for bioinformatics problems. We describe issues with this approach that were not widely known, and introduce our new Burrows Wheeler Markov Distance (BWMD) as an alternative. The BWMD avoids the shortcomings of earlier efforts, and allows us to tackle problems in variable length DNA sequence clustering. BWMD is also more adaptable to other domains, which we demonstrate on malware classification tasks. Unlike other compression-based distance metrics known to us, BWMD works by embedding sequences into a fixed-length feature vector. This allows us to provide significantly improved clustering performance on larger malware corpora, a weakness of prior methods.

1 Introduction

Compression algorithms can be used to measure the similarity between arbitrary sequences with little required domain knowledge or expertise. They have been used in bioinformatics(Mantaci, Restivo, and Sciortino 2008), time series classification and clustering(Keogh, Lonardi, and Ratanamahatana 2004), and malware analysis (Borbely 2015). The bioinformatics and malware analysis domains can be particularly attractive for compression-based similarity measures. Both of these domains involve "short" sequences of tens of thousands of steps, and can often reach 10^8 steps in length. Other machine learning techniques often fail to work when dealing with sequences of such variety and length.

In this work, we note that the Extended Burrows Wheeler Transform (EBWT) (Mantaci et al. 2005) is a compression-based distance metric designed explicitly around the Burrows Wheeler Transform (BWT) (Burrows and Wheeler 1994) algorithm for use in bioinformatics. While EBWT has been useful in that domain, we have discovered a number of weaknesses in this method that reduce its effectiveness and prevent it from being useful in other domains, such as malware detection.

To remedy these issues, we develop a new BWT-inspired distance measure that we refer to as the Burrows Wheeler Markov Distance (BWMD). Unlike EBWT, BWMD is a valid[1] distance metric, and can scale to far larger problems that EBWT cannot tackle due to computational limits. Compared to other compression-based distances, our BWMD is the first to work by embedding a sequence into a Euclidean vector space. This gives a significant advantage to our approach in terms of clustering and query speed. This advantage is achieved by using algorithms that are designed around Euclidean distance, like k-means, that other methods cannot leverage.

We will begin by reviewing related work in the compression distance space, and the needed details of the BWT, the prior method EBWT, and a related method known as LZJD, in Section 2. Next we will begin with a description of the new BWMD in Section 3. In Section 4 we will develop a number of new theoretical insights, proving 1) how EBWT has dramatic failure cases that violate our intuition of how a distance measure should work, 2) that BWMD does not have these failure cases, and 3) comparing how EBWT, BWMD, and LZJD handle randomness, and 4) that BWMD has unique properties in this regard. We will then move into empirical results in Section 5 by comparing BWMD with EBWT on DNA sequence clustering, where we show that BWMD is able to cluster DNA sequences of varying lengths that EBWT fails to cluster in a meaningful way. In Section 6 we will show how BWMD is able to scale to malware classification and clustering tasks that are beyond EBWT's computational ability. Though LZJD provides better classification accuracy at this task, BWMD provides superior clustering results. Finally we will conclude in Section 7

2 Compression Distances

Compression in general can be seen as one way of performing many machine learning tasks, and has deep connections to statistical methods. Following this intuition, Li et al. (2004) introduced the Normalized Information Distance (NID) as a method of measuring similarity using compression. Given a function $K(x)$ that computes the Kolmogorov complexity (i.e., return the length of the shortest computer program that produces x as output), and the associated conditional Kolmogorov complexity $K(x|y)$ (i.e., the length of the shortest computer program that produces x as output given y as in-

[1]A distance metric is considered true, or valid, if it adheres to the properties of reflexivity, symmetry, and triangularity.

put), the NID is a metric as defined in (1). The Kolmogorov complexities are uncomputable functions, making NID of no practical use.

$$\text{NID}(x, y) = \frac{\max\left(K(x|y), K(y|x)\right)}{\max\left(K(x), K(y)\right)} \tag{1}$$

To remedy this situation, Li et al. (2004) went on to present the Normalized Compression Distance (NCD), which replaces the uncomputable $K(x)$ with $C(x)$, which returns the length of x in bytes after running a compression algorithm. To approximate $K(x|y)$, the concatenation of x and y (denoted by $x\|y$) is used, giving the final form of NCD in (2). The compression algorithm chosen impacts the quality of the results. The bzip and LZMA algorithms have been popular due to a combination of reasonable run time performance and generally satisfactory compression ratios.

$$\text{NCD}(x, y) = \frac{C(x\|y) - \min\left(C(x), C(y)\right)}{\max\left(C(x), C(y)\right)} \tag{2}$$

Where $C(x)$ is the (integer) length of object x when compressed. NCD, and compression-based distances in general, do not require significant feature engineering to be applied in practice. This has made them popular for genomic phylogeny (Li et al. 2004; Cilibrasi and Vitanyi 2005) and malware analysis (Bayer et al. 2009; Apel, Bockermann, and Meier 2009; Bailey et al. 2007; Karim et al. 2005), where sequences are longer than what most other techniques can handle ($10^4 - 10^8$ steps in length), and it can be difficult to extract more sophisticated features manually. However, using compression algorithms naively in NCD leads to difficulties with computational scalability and reduced accuracy/failure cases due to the fact that compression algorithms were not designed for similarity analysis (Borbely 2015; Cebrián et al. 2005). For these reasons, some have looked at converting known clustering algorithms into explicit distance measures. For example, the Lempel Ziv Jaccard Distance (LZJD) (Raff and Nicholas 2017a) converts the LZMA algorithm into a true distance measure, and for malware classification tasks has superior accuracy and runtime compared to NCD[2].

2.1 Extended Burrows Wheeler Transform

The Burrows Wheeler Transform (BWT) (Burrows and Wheeler 1994) is a core component of the bzip compression algorithm, and has been widely used in information retrieval applications due to its ability to accelerate search queries(Ferragina and Manzini 2005). The BWT takes an input string u of length $n = |u|$, over an alphabet Σ, and produces a new string $u' = bwt(u)$. Through the use of an end-of-file (EoF) marker, the BWT is invertible, so u can be recovered from u' without loss.

BWT's utility in compression is best understood through example. Consider Table 1, where the BWT transform of the string "easypeasy" is shown. BWT adds a special EoF marker "$", and lexicographically sorts every single-character rotation of the string (observe column *First*, which is in sorted order). The BWT output is then the last column of each string, shown in column *Last*. By computing the BWT version of the string, we can see how runs of the same character ("ee", "aa", "ss") have been created that previously did not exist. Simple run-length encoding can then be applied to produce a compressed version of the string.

Table 1: BWT transform of the string "easypeasy". Column *First* shows the first letter that rotations are sorted by, with the next *Rotation* column showing the rotated string, and the *Last* column showing the tail character of each rotation. The *Last* column is from top-down is the BWT encoding order.

First	Rotation	Last
$	$easypeasy	y
a	asy$easype	e
a	asypeasy$e	e
e	easy$easyp	p
e	easypeasy$	$
p	peasy$easy	y
s	sy$easypea	a
s	sypeasy$ea	a
y	y$easypeas	s
y	ypeasy$eas	s

These sorted rotations can be computed in $\mathcal{O}(n)$ time, and provide a simple method of compression. To turn this into a distance measure, Mantaci et al. (2005) developed the Extended Burrows Wheeler Transform (EBWT). The EBWT works by defining the BWT over a pair of inputs u and v, and computing the sorted order of both sequences. An example of this is shown in Table 2.

Table 2: EBWT computation example

bwt(u=bcaa)	bwt(v=ccbab)	EBWT Merge	Source
a a b c	a b c c b	a a b c	u
a b c a	b a b c c	a b c a	u
b c a a	b c c b a	a b c c b	v
c a a b	c b a b c	b a b c c	v
---	c c b a b	b c a a	u
---	---	b c c b a	v
---	---	c a a b	u
---	---	c b a b c	v
---	---	c c b a b	v

The distance between the two sequences u and v is defined by (3), where rep(i) returns how many times the i'th source occurred in a row, and that only t source transitions occurred.

$$ebwt(u, v) = \sum_{i=1}^{t} \max(\text{rep}(i) - 1, 0) \tag{3}$$

Again, this concept is easier understood through example. Considering Table 2, we can see that the source string sequence is $uuvvuvuvv$. If we group this by transitions, we get $u^2v^2uvuv^2$. Thus the distance $ebwt(u, v) = 1 + 1 + 0 + 0 + 0 + 1 = 3$.

Mantaci et al. (2005) developed the EBWT for applications in bioinformatics, and developed theory to show a num-

[2] A Java (Raff and Nicholas 2018b) and Python (Raff, Aurelio, and Nicholas 2019) implementations of LZJD are available.

ber of situations under which the EBWT will perform well or have desirable properties. However, it is still expensive to compute, requiring $\mathcal{O}(|u| + |v|)$ time for every distance computation. This makes EBWT less attractive as bioinformatic sequences become longer, and reduces its utility in other domains in which compression distances have found use, such as malware classification. While it was known that EBWT did not satisfy the triangle inequality, preventing it from being a true distance metric, previously unreported theoretical issues also exist. We will discuss these issues in 4.

3 Burrows Wheeler Markov Distance

Inspired by the prior work we have just discussed, we now develop a new distance measure based on the Burrows Wheeler Transform. We will refer to our method as the Burrows Wheeler Markov Distance (BWMD), and it is simple to implement. To begin, consider again Table 1, where BWT("easypeasy") is shown. The BWT's effectiveness as a compression algorithm comes explicitly from its ability to re-order the content such that repetitions are reduced to a first-order occurrence. This is why run-length encoding is effective. Because first-order compression is independently effective after the BWT, we do not need to consider more complex interactions over the extent of a file.

We also do not care about the invertibility of any transform. Our goal is to define a new feature space where we can perform effective machine learning and information retrieval. So we seek to build a small statistical summary of the data, rather than build an object from which we can recreate the original data. By measuring the similarity of these small summaries, we measure the similarity of the underlying sequences. Given that first-order compression is effective with the BWT, we chose to select a first-order statistical model. In particular, we can use a Markov model of the probability of observing token u'_i given the previous token u'_{i-1}. The transition matrix $T \in \mathbb{R}^{|\Sigma|\times|\Sigma|}$ can then be used as a statistical summary of the entire sequence BWT(u).

This is all that is needed to describe BWMD, and a succinct description is given below. Each step takes $\mathcal{O}(n)$ time for an input sequence of n bytes. $\mathbb{1}[z]$ is the indicator function, which returns 1 if z is true, and 0 otherwise, and α, β are the rows and columns of the transition matrix.

1. For each sequence u in a corpus of size N
2. Compute $u' = \text{BWT}(u)$
3. Estimate flattened Markov transition vector

$$x[\alpha + \beta \cdot |\Sigma|] = \frac{1}{|u'| - 1} \sum_{i=2}^{|u'|} \mathbb{1}\left[u'_i = \alpha \wedge u'_{i-1} = \beta\right]$$

4. Normalize x such that $x[i] = \sqrt{x[i]}/\sqrt{2}$.

After step (4) in the above process, we obtain from the input u a single feature vector x which we might use, in place of u, in machine learning or information retrieval algorithms. Regardless of the length of the input sequence u, the size of the vector x will depend only on the alphabet size $|\Sigma|$. When working on raw bytes, this would be a 256^2 feature vector. While such a vector takes up to 256 KB per sequence u, the individual input data objects we consider in this work range from 1 MB in size up to 400 MB. This makes the BWMD description quite compact by comparison. When u is shorter in length, the vector x can be stored in a sparse form, making the memory cost $\mathcal{O}(\min(|u|, |\Sigma|^2))$

The normalization in step (4) is also chosen intentionally. The Hellinger Distance is a metric over probability distributions. For the case of two discrete probability distributions $P = (p_1, \ldots, p_k)$ and $Q = (q_1, \ldots, q_k)$, the Hellinger Distance is defined as follows:

$$H(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} \left(\sqrt{p_i} - \sqrt{q_i}\right)^2} \tag{4}$$

Due to the form of (4), the Hellinger distance corresponds to the Euclidean distance between two transformed and scaled versions of P and Q. By using the square root of the coefficients in step (4), divided by $\sqrt{2}$, we have a feature vector x that has been placed into a space where the Euclidean distance can be computed as usual. The results are then equivalent to the Hellinger distance between Markov transition vectors, giving us a statistically sound interpretation for BWMD. That the BWMD is a valid distance metric also follows immediately from the use of the Euclidean distance, which is well known to be a valid metric, a property that the EBWT lacks.

We explicitly use the Hellinger distance over other alternatives, like KL-Divergence, because it corresponds exactly to the Euclidean distance after transformation. This makes is a valid distance metric for which we can make mathematical statements about its behavior with little work, and prove it does not have the same shortcomings as prior methods in this space. In addition, most other clustering and fast retrieval algorithms are built around the Euclidean distance, making our method compatible with a maximal number of complimentary techniques. This is not the case for any prior compression based measure. The ability to use algorithms like k-means, where other alternatives cannot, provides BWMD with advantages in terms of clustering accuracy, as well as computational efficiency to handle big data.

Note that because of the BWT, our approach is *not* equivalent to 2-grams. Our comparison with LZJD, which can be interpreted as adaptive variable-length gram(Raff and Nicholas 2017b), and BitShred, which uses 16-grams, will show that our method is meaningfully more effective than simple n-gram approaches.

4 Theoretical Results

We begin by developing a stronger theoretical understanding of our new method, as well as the prior approach EBWT. Prior works have looked at a number of properties of the EBWT(Mantaci et al. 2007; Mantaci, Restivo, and Sciortino 2008; Mantaci et al. 2005), and describe situations in which EBWT will behave as a metric for a subset of possible inputs, and that it is invertible like the standard BWT. However, our interest in BWT is as a general purpose similarity measure for information retrieval and machine learning applications. We begin by showing three undesirable properties of the EBWT that reduce our confidence in its use for such applications.

Then we will investigate the nature of our new BWMD in these same cases where EBWT might fail.

4.1 EBWT Shortcomings

First we show a simple property that is a direct result of the EBWT measuring distance as a function of repeated source sequences. When we have two strings u and v in any alphabet Σ, it is necessarily the case that the distance is bounded below by the difference in sequence lengths $|u|$ and $|v|$. If u and v differ significantly, we are unlikely to be able to make meaningful similarity comparisons.

Theorem 1. *The distance* $ebwt(u, v) \geq ||u| - |v|| - 1$.

Proof. Consider any two strings u and v. The minimum distance involves the maximum number of transitions between string sources. If $|u| < |v|$, that means there can be at most $2 \cdot |u|$ transitions, going back and forth between u and v on the merging. That necessitates $|v| - |u|$ repetitions at the end. Given the definition of $ebwt$ in (3), that means the minimum distance between u and v must be $|v| - |u| - 1$. □

The above result leads us to suspect that EBWT will not be useful when the sequences being compared are of varying lengths. The greater the difference in sequence length, the more troubling this issue might become. Given the insight from 1, we move on to a more serious departure from our intuition of how a distance measure should behave. In particular, if u is a subset of v, we should expect the distance between u and v to be small. Instead, it is possible for EBWT to return the maximal distance under this scenario.

Theorem 2. *It is possible for* $ebwt(u, v) = |v| + |u| - 2$*, the maximum possible distance, even if* $u \subset v$.

Proof. Consider the string $u = a^{n_1}$ and $v = a^{n_2}$, such that $n_2 > n_1$, $|u| = n_1$ and $|v| = n_2$. Because of the topographical sorting, all rotations of u and v will have the same characters, and so the sorting will only resolve once the max substring length is reached. Since all rotations in u are of length n_1, which is shorter than n_2, a sorting will place all rotations of u before any rotations of v. This results in a transition pattern of $u^{n_1}v^{n_2}$, and thus $ebwt(u, v) = n_1 - 1 + n_2 - 1 = |u| + |v| - 2$. □

2 defies our expectations in the case of similar inputs. If we use the behavior of the NID from (1), we would expect the distance in this scenario to be small. Consider the proof's example with $u = a^{n_1}$ and $v = a^{n_2}$: we would expect NID(u,v) $\leq \log(n_2/n_1)$. This is because v could be encoded as the sequence u repeated n_2/n_1 times, which in the worst case, can be represented in a number of bits logarithmic in the value being encoded (i.e., the nature of any big-integer representation is that the maximum value that can be represented grows exponentially with a doubling of the bits).

We now show that EBWT likewise surprises us in the case of dissimilar inputs, where we can have u and v with no overlap in content, but EBWT identifies as having maximal similarity.

Theorem 3. *It is possible for* $ebwt(u, v) = 0$*, even if there exists no shared characters between* u *and* v*. More formally, there exists* u, v *and an alphabet* $|\Sigma| \geq |u| + |v|$ *such that for every* $x \in u$ *and* $z \in v$*,* $x \notin v \wedge z \notin u$ *yet* $ebwt(u, v) = 0$.

Proof. Let $a \| b$ denote the concatenation of a and b, and $\Big\|_{i=1}^{N} i$ the sequential concatenation of $1, 2, \ldots, N$. Without loss of generality, define the string $u = \Big\|_{i=1}^{n_0} 2 \cdot i$ and $v = \Big\|_{i=1}^{n_0} 2 \cdot i + 1$. Thus $|u| = |v|$, but in the lexicographical sorting u and v will alternate between rotations $u[0] = 2$, $v[0] = 3$, $u[1] = 4$, $v[1] = 5$, …, $u[n_0] = 2 \cdot n_0$, $v[n_0] = 2 \cdot n_0 + 1$. Thus $ebwt(u, v)$ will always contain transitions with no source repetition, hence $ebwt(u, v) = 0$. □

The reader may note that the construction of 3 would allow one to argue that the scenario should return a small distance under the ideal Kolmogorov distance with NID. This could be argued because the construction of u and v allows v to be represented as $v = u + 1$. However, the same scenario can occur with randomized strings where the alphabet does not increment with any simple pattern and is filled with random tokens, so long as there is no overlap in the tokens and the tokens "balance out" once sorted (i.e., there is no $f(\cdot)$ s.t. $v[i] = f(u[i])$ yet $v[i] > u[i] \wedge u[i + 1] > v[i] \forall i$). This requires further expanding the alphabet size $|\Sigma|$, and as such makes 3 the least practical of our concerns. However, we find it enlightening as a theoretical shortcoming which we would prefer to avoid.

While these issues give us cause for concern, EBWT has found use in practice. We will show in 5 that our concern for EBWT's utility is more justified when sequences have varying length.

4.2 Behaviors of BWMD

To delve into BWMD's behavior, we will begin by analyzing the same scenarios used to show our theoretical concerns with the EBWT in the preceding section, as well as compare the behavior of BWMD to that of the LZJD algorithm.

Corollary 1. *Given* $u = a^{n_1}$ *and* $v = a^{n_2}$*, such that* $n_2 > n_1$*, then* $bwmd(u, v) = 0$.

Proof. Under the construction of the embedding of u, $x_u \in \mathbb{R}^{|\Sigma|}$, $\|x\|_0 = 1$ since there will be only one transition pattern of $a -> a$. As such, the value at that index i will be $x_u[i] = \sqrt{1}/\sqrt{2} = 1/\sqrt{2}$. Since the embedding x_v will have the same construction, and thus, $x_u[i] - x_v[i] = 0$, and $\forall j \neq i, x_u[j] = x_v[j] = 0$. Therefore, the distance between u and v will be zero. □

The above also does not conform to expectation with respect to the NID, because we are ignoring the length of the inputs in our computation of distances. The NID(u,v) would be greater than zero in this scenario and is necessitated by storing the difference in repetition lengths n_2 and n_1. This tells us BWMD will be less sensitive to differences in sequence length, which may be desirable or not, depending on the application.

The behavior of LZJD in this scenario was used to prove its sensitivity to potential repetition of the input. It was shown

that $\text{LZJD}(a^{n_1}, a^{n_2}) = 1 - \frac{\sqrt{8n_1+1}-1}{\sqrt{8n_2+1}-1}$ as a lower bound for a similar scenario (Raff and Nicholas 2017a). This distance grows at a rate considerably faster than logarithmic, but is also better than the EBWT distance in this case. We conclude that both LZJD and BWMD have better behavior, but BWMD will lower-bound the NID and LZJD will upper-bound the NID.

In a similar manner as 1 was shown, the same construction can be applied to 3's issue for BWMD.

Corollary 2. *For all u, v such that for every $z \in v$, $x \notin v \wedge z \notin u$, then $bwmd(u, v) = 1$, the maximum possible distance.*

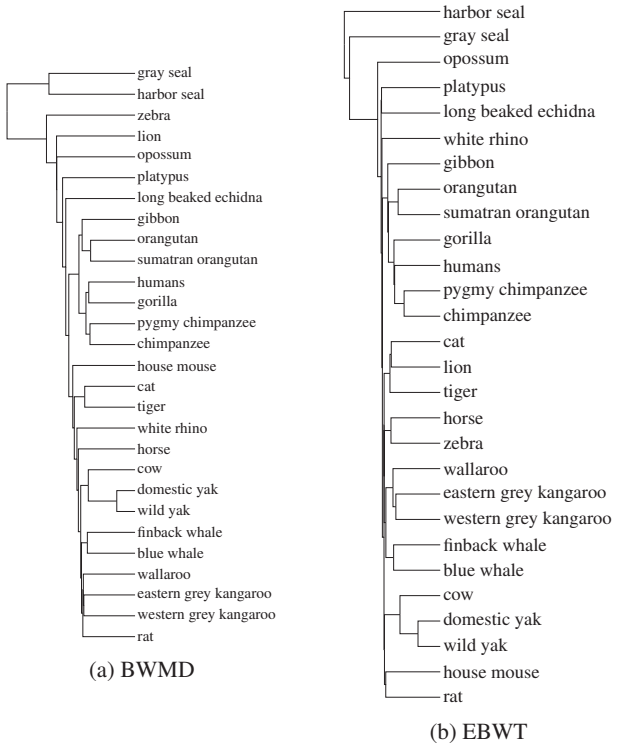
The derivation follows from the fact that $\sum_{i=1}^{k} \sqrt{p_i}^2 = \sum_{i=1}^{k} p_i = 1$. This means the distance computation will reduce to $1/\sqrt{2}\sqrt{2 \cdot 1} = 1$. Therefore, the distance when the embeddings x_u and x_v have no intersection is maximized. In this case, BWMD aligns well with the behavior we would expect from the NID. Likewise it is easy to see that LZJD will return its maximum distance of 1 in this scenario as well. LZJD measures the set intersection, so when the sets have no intersection, then maximal distance is achieved.

5 Genomic Clustering

We begin by showing that our new BWMD has similar utility as the original EBWT distance for genomic phylogeny from DNA sequences. This was the original proposed use of the EBWT measure, where they evaluated the Single-Link Clustering results on mitochondrial DNA (mtDNA) (Mantaci et al. 2005). Such data can be obtained using the NIH GeneBank , which we have used to create a similar corpus of DNA sequences to compare the relative pros and cons of BWMD and EBWT. We will use both mtDNA as has been done in prior work, but also a more challenging case with chromosomal genomic scaffold DNA sequences. We will produce dendrograms for each tasks with Single-Link Clustering (SLINK).

First, we will evaluate mtDNA data on 28 different species, and use Single-Link Clustering to produce a dendrogram of the species based on their mtDNA. The results are shown in 1, with both EBWT and BWMD taking under 1 second to perform the clustering. Our goal is not to fully evaluate the quality of each dendrogram, but to show that both methods produce reasonable results in this case, which may be of interest to researchers in bioinformatics. Both EBWT and BWMD do reasonably well at this task, with differing mistakes, advantages, and disadvantages.

EBWT gets most base level groups correct (e.g., lion, tiger, cat in one group, primates grouped together). There is a failure to properly group the harbor and gray seals as related to each other, and instead act as outliers which SLINK is forced into a cluster at the end at higher cost. EBWT also fails to group the white rhino with other members of the Ferungulates family (e.g., the horse and zebra would have been closest members) (Cao et al. 1998). BWMD was able to correctly pair the seals and placed white rhino with a larger family of Ferungulates (closest to horse, which is correct, and with the cows and yak which are members). But BWMD failed to place the mouse and rat together, and dispersed the



(a) BWMD

(b) EBWT

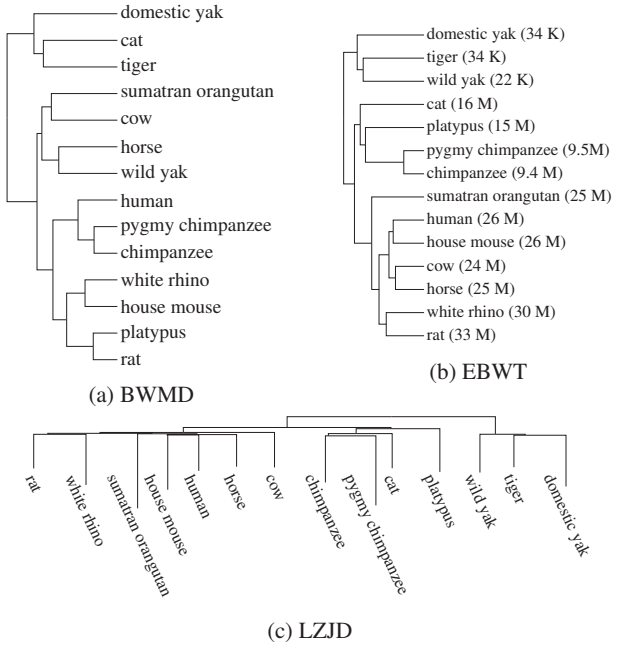
Figure 1: Single-Link Clustering result on mitochondrial mtDNA data.

zebra and lion from their more appropriate neighbors.

Results with both methods are reasonable. However, the mtDNA task is an easier task. All of the mtDNA sequences are of similar sizes, with the western grey kangaroo being shortest at 15 KB in length and the lion being longest at 17 KB. Our theoretical results in 4 would indicate that we may see more significant issues if we had sequences of varying length. We explore this with a dataset of chromosome genomic scaffold DNA for a subset of the species evaluated. We selected one whole scaffold DNA sequence from a random chromosome for the 11 species where this was available. We selected "unplaced genomic scaffold" sequences for 3 remaining species (the yaks and tiger), which is a much shorter and incomplete amount of data. This gives us a minimum sequence size of 22 KB and a maximum of 33 MB.

The SLINK results are shown in Figure 2. At a base level, the cost to perform EBWT and its scalability issues are more pronounced. BWMD takes only 47 seconds to perform SLINK clustering. EBWT took 28 minutes, making it over 35× slower. When plotting the EBWT dendrogram in Figure 2b, we include the size of the DNA sequence in parentheses. When organized in this way, it becomes clear that the EBWT clustering is degenerate, and corresponds exactly to file-size, rather than content.

In contrast, the BWMD in Figure 2a produces reasonable groupings despite disparate sequence lengths. For example, (full scaffold) cat and (unplaced and incomplete) tiger are correctly grouped despite the cat sequence being 450× longer. The BWMD results are not perfect: the orangutan and do-



(a) BWMD

(b) EBWT

(c) LZJD

Figure 2: Single-Link Clustering on Genomic Scaffolding.

mestic yak were placed farther from the other (well-grouped) primates and ferungulates respectively. Overall we can see that groups which should be placed near each other are, without degrading to sequence length information.

We include LZJD (Figure 2c) in this scenario, to demonstrate that BWMD has increased value over other alternatives. Here we can see that LZJD suffers the same failure as EBWT in placing the three smallest partial segments into a single cluster. With the exception of correctly grouping the two chimpanzee species, the LZJD dendrogram is degenerate.

These results are in line with our theory derived in 4. When working with sequences of homogeneous length, EBWT performed well. But BWMD is able to handle disparate sequence lengths reasonably well, where EBWT degrades to grouping by sequence length rather than content.

BWMD's ability to handle the original mtDNA data, as well as substantially better results with the irregular-sized scaffold DNA, is made more impressive by the fact that BWMD is encoding everything into $\mathbb{R}^{16}$ due to the small alphabet $|\Sigma| = 4$. This is a reduction in storage cost by a factor of up to 515.6×, and allows for more flexibility in creating a larger and searchable index using BWMD.

5.1 A note on BMWD's Disadvantage

It is also worth noting that, from an information encoding perspective, BWMD is at a disadvantage in this testing over DNA data. EBWT is dimensionless, and has the representation capacity of the merged sorting of two different strings, meaning its representational capacity is a function of the sequence length under consideration. BWMD encodes each sequence into a fixed-length feature vector of size $|\Sigma|^2$. Since we are working with DNA data, the alphabet $\Sigma = \{A, T, C, G\}$ is quite small. As such all DNA sequences in these experiments, up to 33 MB in size, are being embedded into a 16-dimensional space. BWMD's ability to match or outperform EBWT means we must be doing a significantly better job at leveraging the first-order information expressed by the Burrows Wheeler Transform.

6 Malware Results

It can be difficult to reliably parse malicious software as malware authors may intentionally violate rules and format standards. Compression based similarity measures are useful in this case as they allow us to avoid these complex parsing issues. In this section we will look at the classification accuracy of BWMD and LZJD for several datasets. Our expectation is that LZJD will have better classification accuracy, due to Lempel-Ziv compressors being more effective than those based on Burrows Wheeler. However, we find that BWMD has significant advantages when clustering is the goal accuracy by up to 65.6×, and is 24× faster to search large corpora by obtaining sub-linear scaling with no loss in accuracy. Since LZJD cannot achieve this, this advantage will only increase with corpus size.

Many prior works have looked at malware classification and clustering by processing raw bytes, due to the difficulty of parsing malware. We will compare with the seminal BitShred algorithm (Jang, Brumley, and Venkataraman 2011) for clustering. Other similar byte based distance functions such as Ssdeep and Sdhash where evaluated, but founds to have degenerate performance on our smallest and easiest corpora, which can be found in appendix section 6.1. Compression distances such as EBWT and NCD are not evaluated in this section, because it would require multiple compute months for our smallest dataset, and simply cannot scale to the size of our malware corpora.

Table 3: Malware datasets used in experiments.

Dataset	Avg Size	# Classes	Train	Test	Storage Size
EMBER	1.17 MB	2	600,000	200,000	936 GB
VS20F	705 KB	20	160,000	40,000	141 GB
Kaggle Bytes	4.67 MB	9	10,868	—	50.8 GB
Kaggle ASM	13.5 MB	9	10,868	—	147 GB
Drebin APK	1.37 MB	20	4,664	—	6.4 GB
Drebin TAR	1.84 MB	20	4,664	—	8.6 GB

For our evaluation, we will use several datasets summarized in Table 3. The EMBER dataset (Anderson and Roth 2018) pertains to a binary classification problem of "benign vs malicious" for Windows executables. Because there are only two classes, clustering results will not be evaluated on this corpus. However it is by far the largest corpus, allowing us to explore the scalability of our algorithms. The raw files can be obtained from VirusTotal (www.virustotal.com) and are nearly 1TB total. Our remaining datasets will be multi-class problems where each sample is a member of a specific malware family. We will use these to evaluate both classification accuracy, as well as accuracy in clustering with respect to the class labels. Using VirusShare (Roberts 2011) we create another Windows based dataset with 20 malware families. The families were determined using VirusTotal and the AV-

Class tool which determines a single canonical malware family label based on multiple Anti-Virus outputs (Sebastián et al. 2016). We select the 20 most populous families, and use 7,000 examples for training and 3,000 for testing. The last four datasets we use are evaluated in "two forms", following prior work (Raff and Nicholas 2017a). The Kaggle datasets are from a 2015 Kaggle competition sponsored by Microsoft (Ronen et al. 2018). In the "Bytes" version our algorithms are run on the raw malware binary, and in the "ASM" version the output of IDA-Pro's disassembler is used instead. From the Drebin corpus (Arp et al. 2014) we use the 20 most populous families, where the "APK" version is the raw bytes of the Android APK (essentially a Zip file with light compression), and the "TAR" version which unpacks the APK and recombines all content into a single tar file.

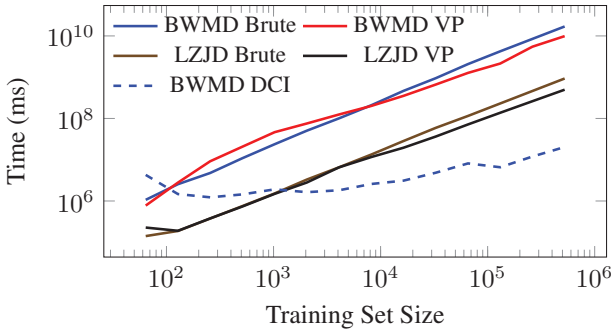


Figure 3: Table shows 9-NN search retrieval speed on the Ember test set (in milliseconds, y-axis, log-scale) as the number of training points (x-axis, log-scale) increases.

6.1 Malware Classification

We begin our analysis by looking at nearest neighbor classification performance of various methods. The performance of each algorithm under this scenario gives us insight not only to its utility, but how effective it would be for analysts in finding similar malware. Utility in this scenario requires both high accuracy *and* computational efficiency, as malware corpora are often measured in the terabyte to petabyte range.

Small Scale Malware Classification The Kaggle and Drebin corpora are considerably smaller in size, allowing us to test a wider selection of methods against them. In the below table we use balanced accuracy, where the weights of each file are adjusted so that the total weight of each class is equal, because malware families are not evenly distributed in each corpus.

We can see from these results that the compression-based approaches, LZJD and BWMD, generally outperform other alternatives by a wide margin. As was expected, LZJD has higher accuracy that BWMD, since LZJD is based on a more effective compression algorithm. While this is a slight weakness of BWMD, its advantage comes in being orders of magnitude faster, as we will show in the large scale testing in the next section. This makes it the only method usable for larger-scale corpora. This also shows that Ssdeep and Sdhash are simply not accurate enough to be considered for use, without regard to computational constraints.

BWMD performed second best on every dataset, the only exception being a 3 point difference to the BitShred algorithm. However, BWMD outperformed BitShred by at least 11 points on all other datasets. Supporting our theoretical analysis in 4, we also see hints that BWMD is better equipped to work with extremely long sequences. Most notably, BWMD is the only method which had improved accuracy and reduced variance when moving from Kaggle Bytes (4.67 MB) to Kaggle ASM (13.5 MB). This suggests that the disassembly may be in a form that allows the BWT to better capture first-order dependencies for compression.

The fact that BWMD has non-trivial accuracy on Drebin APK (random guessing is 5%) is particularly impressive and worth noting. This is because the APK files are essentially Zip files with a standard structure, and the Zip compression format is a more effective one than most BWT based methods such as bzip. As such, that there is any first-order information exploitable for effective similarity search is impressive, and indicates the utility of BWMD in wider applications.

Large Scale Malware Classification On EMBER we use 9-Nearest Neighbors as our classifier so that we can compute meaningful values for the Area Under the ROC Curve (AUC) (Bradley 1997). In this malware detection context, AUC can be interpreted as an algorithm's ability to properly prioritize all malicious software above all benign software. This metric is useful for prioritizing work queues, and is therefore particularly pertinent. We evaluate only BWMD and LZJD due to computational constraints on this larger dataset.

BWMD obtains an AUC of 98.3%, where LZJD acheives a slightly better 99.7% AUC. As expected, LZJD obtains a higher accuracy than BWMD, because LZJD is built upon a more effective compression algorithm. However, accuracy in isolation does not determine what method is best to use. Due to the large size of malware corpora, sub-linear scaling is needed to be useful for realistic sized datasets.

BWMD does have an advantage over LZJD in its ability to scale to large corpora in an efficient manner. This is critical, since industry datasets routinely require comparisons to terabytes of files or more (Roussev 2010). Prior work has tried, with limited success, to scale LZJD to larger corpora. Using an extension of the Vantage Point (VP) tree (Yianilos 1993), only a 2.5x speedup over brute force search was achieved(Raff and Nicholas 2018a). Because BWMD operates by embedding files into a Euclidean space, we can leverage specialized algorithms like the Dynamic Continuous Index (DCI) algorithm (Li and Malik 2017) that only work for the Euclidean distance. DCI works by projecting the whole dataset down to different, random, embeddings, and allows obtaining the true nearest neighbors in a fast and efficient manner. LZJD is not compatible with such algorithms, resulting in BWMD being better equipped for this task.

In 3 we compare the total query time of BWMD and LZJD under different indices, as the training set size increases from 64 files up to the full 600,000. We found the VP tree of minimum variance (Raff and Nicholas 2018a) performed best compared to other algorithms like KD and Cover-trees, and so only its results are included. In the dashed line we show

Table 4: Balanced Accuracy results for 1-NN classification on each dataset. Results show mean 10-Fold Cross Validation accuracy (standard deviation in parentheses). Best results in **bold**, second best in *italics*.

Dataset	Ssdeep	Sdhash	BitShred	BWMD	LZJD
Kaggle Bytes	38.4 (1.4)	60.2 (2.3)	43.7 (1.9)	*96.4 (2.2)*	**97.9 (1.4)**
Kaggle ASM	26.6 (2.2)	28.8 (1.3)	36.9 (1.6)	*97.0 (1.8)*	**97.1 (1.7)**
Drebin APK	13.6 (1.6)	5.8 (0.5)	*58.3 (3.9)*	55.3 (4.4)	**81.0 (4.0)**
Drebin TAR	24.2 (2.9)	8.3 (1.2)	65.1 (3.7)	*76.3 (3.6)*	**87.9 (2.1)**

Table 5: Clustering performance of BWMD, LZJD, and BitShred. Best results shown in **bold**.

		$k = C$			$k = 10 \cdot C$		
Dataset	Metric	BWMD	LZJD	BitShred	BWMD	LZJD	BitShred
Kaggle Bytes	V-M	**0.581**	0.352	0.007	**0.546**	0.414	0.028
	Homog	**0.597**	0.254	0.003	**0.885**	0.378	0.015
	Complt	0.566	**0.573**	0.239	0.396	**0.457**	0.265
Kaggle ASM	V-M	**0.528**	0.235	0.014	**0.562**	0.531	0.366
	Homog	**0.550**	0.176	0.007	**0.911**	0.599	0.291
	Complt	**0.508**	0.351	0.257	0.407	0.477	**0.495**
Drebin APK	V-M	**0.307**	0.219	0.095	**0.412**	0.326	0.389
	Homog	**0.296**	0.172	0.054	**0.566**	0.313	0.333
	Complt	0.319	0.303	**0.375**	0.323	0.340	**0.468**
Drebin TAR	V-M	**0.403**	0.248	0.065	**0.508**	0.478	0.386
	Homog	**0.416**	0.177	0.036	**0.754**	0.503	0.332
	Complt	0.391	**0.413**	0.335	0.383	0.455	**0.460**
VS20F	V-M	**0.353**	0.009	0.009	**0.449**	0.204	0.056
	Homog	**0.328**	0.005	0.005	**0.562**	0.137	0.030
	Complt	**0.381**	0.249	0.221	0.374	**0.400**	0.378

BWMD accelerated with the Dynamic Continuous Index (DCI) algorithm (Li and Malik 2017).

We can see that as the training corpus becomes larger, the VP trees are able to get small constant factor speedups, but are not able to reliably prune large portions of the search space. Because BWMD is in Euclidean space, it is the only method able to leverage the DCI algorithm and thus able to get significant order-of-magnitude search speedups. This combination makes BWMD 24× faster than LZJD (5.6 CPU hours compared to *5.6 days*), and 834× faster than BWMD with a brute force search. One can clearly see that DCI's scaling is sub-linear, and its advantage grows with the corpus size. This is obtained with no loss in accuracy on the Ember corpus, making BWMD the only effective approach for scaling to even larger corpora.

6.2 Malware Clustering

In this section we will show that BWMD has significant advantages in terms of clustering malware into families. This benefit comes largely from BWMD mapping sequences into a Euclidean feature space, where we can leverage tried-and-true algorithms like k-means to perform fast and useful clustering. LZJD is incompatible with k-means, and similar methods that require an explicit euclidean feature vector. As such LZJD, like BitShred, is constrained to distance based clustering methods like agglomerative clustering. This puts them at a significant disadvantage compared to BWMD.

Evaluating the quality of our clustering results, we will consider three measures: Homogeneity, Completeness, and V-Measure, as introduced by Rosenberg and Hirschberg (2007) and using the class labels as ground truth cluster assignments. Homogeneity measures how well an algorithm does at making each found cluster as "pure" as possible (i.e., only one class in each cluster). Completeness measures how well an algorithm groups all examples of a class into as few clusters as possible (i.e., all examples of one class in only one cluster). V-Measure is the harmonic average of Homogeneity and Completeness. All three metrics are measured on the scale $[0, 1]$, with 0 being worst, and 1 being the maximum score.

In performing the clustering, we will test using $k =$ the true number of classes and $k = 10\times$ the true number of classes. The former ($k = C$) is done to judge how well the clustering algorithms are able to recover the underlying ground truth. The latter ($k = 10 \cdot C$) is done as it corresponds best to how a malware analyst would desire to use these tools.It is easier to over-estimate the number of clusters than to predict the exact value of k, and by clustering an analyst would hope to reduce their workload by quickly checking that files in the same cluster are related, and then performing an in-depth analysis on only a few representatives from each cluster (VanHoudnos et al. 2017). For this reason, we consider Homogeneity the most important of the three measures, as it corresponds with how an analyst would use clustering, followed by V-Measure, and then Completeness.

BWMD is the only method that can leverage the k-Means algorithm, and we use Hamerly's variant because

it avoids redundant computation while returning the exact same results(Hamerly 2010). For LZJD and BitShred we use Average-Link clustering using a fast $\mathcal{O}(n^2)$ algorithm (Müllner 2011). While the original BitShred paper used Single-Link, we found Average link provided the best results across all metrics for both BitShred and LZJD. The results are shown in Table 5, where we can see BWMD dominates LZJD and BitShred by our most important metrics, Homogeneity and V-Measure[3]. BWMD's advantage in this regard is often dramatic. For example, BWMD scores $2.34\times$ better on Homogeneity compared to LZJD when $k = 10 \cdot C$ on the Kaggle bytes dataset, and $59\times$ better than BitShred. While BWMD does not always perform best by the Completeness metric, it is always competitive with the best scoring method, which is why BWMD dominates by V-Measure . The results overall clearly indicate that BWMD provides the best clusterings across multiple datasets, of different encodings, and different numbers of clusters, showing the flexibility of the compression-based approach.

Because BWMD can leverage the k-means concept and the many efficient algorithms for its computation, it is also the most scalable for these methods. LZJD and BitShred are inherently limited by the $\mathcal{O}(n^2)$ lower-bound complexity of hierarchical clustering [4] . For example, BWMD took only 27 minutes to over-cluster the 160,000 files in the VS20F training set, the largest under consideration. This is $17.3\times$ faster than LZJD which took 7.76 hours, and $54.6\times$ faster than Bitshred at just over a day.

7 Conclusion

We have developed and introduced the Burrows Wheeler Markov Distance (BWMD), a new distance metric inspired by the Burrow Wheeler Transform.

A theoretical analysis has shown several ways in which BWMD has better behavior, which is confirmed by showing new abilities for clustering DNA sequences that prior methods could not handle. For malware clustering, we have shown BWMD considerably outperforms prior methods in both speed and accuracy, and BWMD is the only byte-based method which can achieve sub-linear search scaling on larger corpora.

References

Anderson, H. S., and Roth, P. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*.

Apel, M.; Bockermann, C.; and Meier, M. 2009. Measuring similarity of malware behavior. In *2009 IEEE 34th Conference on Local Computer Networks*, number October, 891–898. IEEE.

Arp, D.; Spreitzenbarth, M.; Malte, H.; Gascon, H.; and Rieck, K. 2014. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. *Symposium on Network and Distributed System Security (NDSS)* (February):23–26.

Bailey, M.; Oberheide, J.; Andersen, J.; Mao, Z. M.; Jahanian, F.; and Nazario, J. 2007. Automated Classification and Analysis of Internet Malware. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, RAID'07, 178–197. Berlin, Heidelberg: Springer-Verlag.

Bayer, U.; Comparetti, P. M.; Hlauschek, C.; Kruegel, C.; and Kirda, E. 2009. Scalable , Behavior-Based Malware Clustering. *NDSS* 9.

Borbely, R. S. 2015. On normalized compression distance and large malware. *Journal of Computer Virology and Hacking Techniques* 1–8.

Bradley, A. P. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* 30(7):1145–1159.

Burrows, M., and Wheeler, D. J. 1994. A Block-sorting Lossless Data Compression Algorithm. Technical report, DEC Systems Research Center, Palo Alto, Califorina.

Cao, Y.; Janke, A.; Waddell, P. J.; Westerman, M.; Takenaka, O.; Murata, S.; Okada, N.; Pääbo, S.; and Hasegawa, M. 1998. Conflict Among Individual Mitochondrial Proteins in Resolving the Phylogeny of Eutherian Orders. *Journal of Molecular Evolution* 47(3):307–322.

Cebrián, M.; Alfonseca, M.; Ortega, A.; and others. 2005. Common pitfalls using the normalized compression distance: What to watch out for in a compressor. *Communications in Information & Systems* 5(4):367–384.

Cilibrasi, R., and Vitanyi, P. M. B. 2005. Clustering by Compression. *IEEE Transactions on Information Theory* 51(4):1523–1545.

Ferragina, P., and Manzini, G. 2005. Indexing Compressed Text. *J. ACM* 52(4):552–581.

Hamerly, G. 2010. Making k-means even faster. In *SIAM International Conference on Data Mining (SDM)*, 130–140.

Jang, J.; Brumley, D.; and Venkataraman, S. 2011. BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis. In *Proceedings of the 18th ACM conference on Computer and communications security - CCS*, 309–320. New York, New York, USA: ACM Press.

Karim, M. E.; Walenstein, A.; Lakhotia, A.; and Parida, L. 2005. Malware phylogeny generation using permutations of code. *Journal in Computer Virology* 1(1):13–23.

Keogh, E.; Lonardi, S.; and Ratanamahatana, C. A. 2004. Towards Parameter-free Data Mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, 206–215. New York, NY, USA: ACM.

Li, K., and Malik, J. 2017. Fast k-Nearest Neighbour Search via Prioritized DCI. In *Thirty-fourth International Conference on Machine Learning (ICML)*.

Li, M.; Chen, X.; Li, X.; Ma, B.; and Vitanyi, P. M. 2004. The Similarity Metric. *IEEE Transactions on Information Theory* 50(12):3250–3264.

[3]Not included due to space limitations, BWMD also dominates by Normalized Mutual Information and Adjusted Rand Index.

[4]Other alternatives, like k-medoids, have similar $\mathcal{O}((n-k)^2k)$ complexity.

Mantaci, S.; Restivo, A.; Rosone, G.; and Sciortino, M. 2005. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In *Proceedings of the 16th Annual Conference on Combinatorial Pattern Matching*, CPM'05, 178–189. Berlin, Heidelberg: Springer-Verlag.

Mantaci, S.; Restivo, A.; Rosone, G.; and Sciortino, M. 2007. An extension of the Burrows–Wheeler Transform. *Theoretical Computer Science* 387(3):298–312.

Mantaci, S.; Restivo, A.; and Sciortino, M. 2008. Distance measures for biological sequences: Some recent approaches. *International Journal of Approximate Reasoning* 47(1):109–124.

Müllner, D. 2011. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*.

Raff, E., and Nicholas, C. 2017a. An Alternative to NCD for Large Sequences, Lempel-Ziv Jaccard Distance. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*, 1007–1015. New York, New York, USA: ACM Press.

Raff, E., and Nicholas, C. 2017b. Malware Classification and Class Imbalance via Stochastic Hashed LZJD. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, 111–120. New York, NY, USA: ACM.

Raff, E., and Nicholas, C. 2018a. Toward Metric Indexes for Incremental Insertion and Querying. *arXiv*.

Raff, E., and Nicholas, C. K. 2018b. Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash. *Digital Investigation*.

Raff, E.; Aurelio, J.; and Nicholas, C. 2019. PyLZJD: An Easy to Use Tool for Machine Learning. In Calloway, C.; Lippa, D.; Niederhut, D.; and Shupe, D., eds., *Proceedings of the 18th Python in Science Conference*, 97–102.

Roberts, J.-M. 2011. Virus Share.

Ronen, R.; Radu, M.; Feuerstein, C.; Yom-Tov, E.; and Ahmadi, M. 2018. Microsoft Malware Classification Challenge.

Rosenberg, A., and Hirschberg, J. 2007. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning(EMNLP-CoNLL)*, 410–420.

Roussev, V. 2010. Data Fingerprinting with Similarity Digests. In Chow, K.-P., and Shenoi, S., eds., *Advances in Digital Forensics VI: Sixth IFIP WG 11.9 International Conference on Digital Forensics, Hong Kong, China, January 4-6, 2010, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg. 207–226.

Sebastián, M.; Rivera, R.; Kotzias, P.; and Caballero, J. 2016. AVclass: A Tool for Massive Malware Labeling. In Monrose, F.; Dacier, M.; Blanc, G.; and Garcia-Alfaro, J., eds., *Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016*. Paris, France: Springer International Publishing. 230–253.

VanHoudnos, N.; Casey, W.; French, D.; Lindauer, B.; Kanal, E.; Wright, E.; Woods, B.; Moon, S.; Jansen, P.; and Carbonell, J. 2017. This Malware Looks Familiar: Laymen Identify Malware Run-time Similarity with Chernoff faces and Stick Figures. In *10th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*.

Yianilos, P. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, 311–321. Society for Industrial and Applied Mathematics.