Guilt by Association: Large Scale Malware Detection by Mining File-relation Graphs

Acar Tamersoy
College of Computing
Georgia Institute of Technology
tamersoy@gatech.edu

Kevin Roundy
Symantec Research Labs
kevin_roundy@symantec.com

Duen Horng Chau
College of Computing
Georgia Institute of Technology
polo@gatech.edu

ABSTRACT

The increasing sophistication of malicious software calls for new defensive techniques that are harder to evade, and are capable of protecting users against novel threats. We present Aesop, a scalable algorithm that identifies malicious executable files by applying Aesop's moral that "a man is known by the company he keeps." We use a large dataset voluntarily contributed by the members of Norton Community Watch, consisting of partial lists of the files that exist on their machines, to identify close relationships between files that often appear together on machines. Aesop leverages locality-sensitive hashing to measure the strength of these inter-file relationships to construct a graph, on which it performs large scale inference by propagating information from the labeled files (as benign or malicious) to the preponderance of unlabeled files. Aesop attained early labeling of 99% of benign files and 79% of malicious files, over a week before they are labeled by the state-of-the-art techniques, with a 0.9961 true positive rate at flagging malware, at 0.0001 false positive rate.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining; D.4.6 [**Security and Protection**]

Keywords

Malware detection; graph mining; file graph; belief propagation; locality sensitive hashing

1. INTRODUCTION

Protection against novel malware attacks, also known as 0-day malware, is becoming increasingly important as the cost of these attacks increases. For individuals, the dollars and cents cost is rising due to the increasing prevalence of financial fraud and the increasing viciousness of malware, such as the CryptoLocker ransomware program that encrypts personal data files and holds them for a ransom of 300 dollars [4]. Emotional and professional costs can be much higher, as when attacks result in the loss of privacy. The situation is arguably worse for governments and businesses, which find themselves under siege by well-funded attackers that routinely create devastating financial losses, and perhaps even more impactful losses of intellectual property and operational secrets [23].

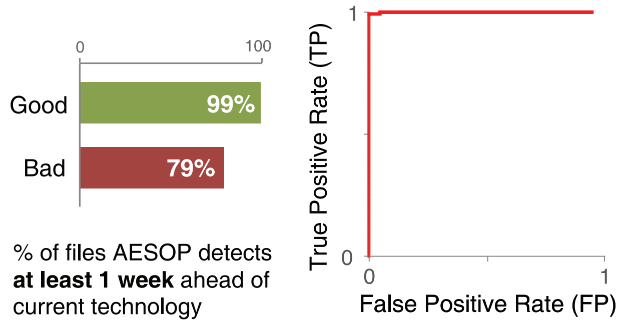

Figure 1: Left: 99% of the known good files and 79% of known bad files detected by Aesop were labeled *at least 1 week* ahead of Symantec's current technology. Right: Aesop achieves almost perfect detection for malware, with few false alarms (0.9961 TP rate at 0.0001 FP rate).

Computer security providers recognize the need to respond with better protection against novel threats. The goal of these 0-day threat protections is to limit the malware's window of effectiveness, so that malicious files are detected as soon as possible after their first appearance. Another critical measure of success is a vanishingly small false positive rate, as labeling a benign file as malicious can have devastating consequences, particularly if it is a popular file or one that is essential to the stability of the system, as in the case of operating system and driver files.

We present Aesop (Figure 2), a novel approach to detecting malicious executable files by applying the well-known aphorism that "a man is known by the company he keeps," and in our case, a file's goodness may be judged by the other files that often appear with it on users' machines. More precisely, we infer unlabeled files' reputation (or goodness) by analyzing their relations with labeled peers.

Aesop is not the first attempt to detect malware by establishing file reputation scores. A representative work in this space is Polonium [7], which leverages the insight that some computer users have poor internet hygiene in that they attract many more malicious files than users that follow se-

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623342.

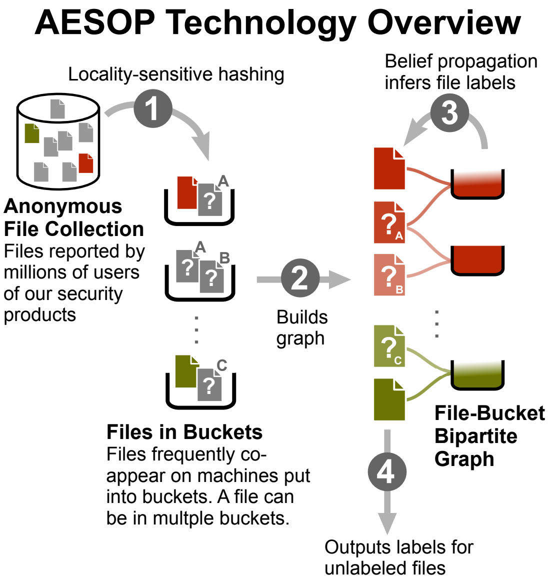
AESOP Technology Overview
Locality-sensitive hashing
1
Anonymous File Collection
Files reported by millions of users of our security products
A
A
B
C
2
Builds graph
Files in Buckets
Files frequently co-appear on machines put into buckets. A file can be in multple buckets.
Belief propagation infers file labels
3
A
B
C
File-Bucket Bipartite Graph
4
Outputs labels for unlabeled files

Figure 2: Overview of the Aesop technology

curity best practices. Polonium constructs a bipartite graph between files and machines, in which a file-machine edge represents the existence of a particular file on a particular machine. This approach proved to be successful, Symantec has deployed Polonium; it has detected millions of malicious files. However, Polonium misses many malicious files as it can only observe malware's file-to-file relationships indirectly through the lens of low-hygiene machines. By contrast, Aesop directly captures file-to-file affinity and can therefore identify malicious files that co-occur with one another, even when they do not appear on heavily infected machines. As we shall demonstrate, Aesop is able to detect many malicious files over a week before they are labeled by Symantec's existing Polonium-based technology, with a 0.0001 false positive rate (see Figure 1).

Like Polonium, in this work we leverage Symantec's Norton Community Watch data, the most important elements of which are unique file and machine identifiers. File identifiers are SHA-256 or MD5 cryptographic hash values that are computed over the file's raw bytes. Symantec's proxy for a true machine identifier is based on the serial number of Norton security products, which is an adequate but imperfect fit because product re-installation on a single machine may result in a serial number change, and a single serial number can be carried from one machine to another. The scale of this dataset is impressive, comprising 119 million machines and 10.1 billion files.

This paper makes the following contributions:

- We formulate the malware detection problem as a large-scale graph mining and inference problem, where our goal is to identify an unknown file's relations with other files so that we can establish guilt or innocence by its association with files that are known to be benign or malicious.
- We present the Aesop algorithm that leverages locality-sensitive hashing to efficiently compute file similarity values to construct a file-relation graph for inferring file goodness based on belief propagation.
- Aesop achieved early detection of 99% of benign files and 79% of malicious files that remained unlabeled by Symantec for over a week before they were eventually labeled, with exceptionally low error rates (see Figure 1).

The remainder of this paper proceeds as follows. We begin by describing our dataset and the notation we will use throughout this paper. We then proceed to a description of Aesop and its various components, followed by the experiments we conducted to demonstrate its effectiveness. Finally, we discuss our plans to deploy Aesop in support of Symantec's malware detection capabilities, and end by presenting our conclusions.

2. NOTATION AND DATASET DESCRIPTION

In this section, we formally describe our dataset and the central notion of file co-occurrence strength.

2.1 Dataset

Our dataset D consists of records of the form $< f, M_f >$, where f is a file and M_f is the set of machines that file f appears on (i.e., M being the set of all the machines, $M_f = \{m_1, m_2, ...\}$ where $m_i \in M$). $|M_f|$ denotes the prevalence of file f, i.e., the number of machines it appears on. Each file is either *labeled* or *unlabeled*. The possible labels for a labeled file are *good* and *bad*, indicating the nature of the file, i.e., whether it is purely trusted or malicious, respectively. We refer to a labeled file with the label *good* as a *good file* and with the label *bad* as a *bad file*. The good and bad files comprise the ground-truth set. Our informal high-level problem statement can be stated as follows: *Given a dataset as defined above, assign a label (i.e., good or bad) to unlabeled files based on their co-occurrence with labeled files.*

2.2 Defining File Co-occurence Strength

We define the *strength* of co-occurrence between two files f_i and f_j based on the overlap between sets M_{f_i} and M_{f_j}, and employ the Jaccard similarity measure given by the formula $J(M_{f_i}, M_{f_j}) = \frac{|M_{f_i} \cap M_{f_j}|}{|M_{f_i} \cup M_{f_j}|}$. This measure takes values between 0 and 1 (inclusive); between files f_i and f_j, the former indicates no co-occurrence relationship and the latter indicates a perfect co-occurrence relationship.

We assume that if $J(M_{f_i}, M_{f_j}) \leq \delta_J$, this indicates a *weak* (not *strong*) co-occurrence between files f_i and f_j. Additionally, if file f has a prevalence less than or equal to δ_M, (i.e., $|M_f| \leq \delta_M$), we call file f an immature file since it did not have time to establish presence, and we deem any co-occurrence relationship it is involved in as weak. Both δ_J and δ_M must be determined based on domain knowledge—for our dataset, we set $\delta_J = 0.5$ and $\delta_M = 4$.[1]

Aesop leverages strong co-occurrence relationships between the files to label them. To achieve this, Aesop uses Jaccard similarity because it can be efficiently computed and well-approximated for our large scale dataset through locality-sensitive hashing, which we will describe in the following sections.

Table 1 lists the symbols used throughout the paper.

[1] Specifically, we set δ_M based on our experience from Polonium [7]. The study on the effects of varying δ_J is omitted due to space constraints.

Symbol	Meaning
f	File (a.k.a. executable, software, application)
m	Machine (or computer)
M_f	The set of machines that file f appears on
D	Input dataset; records consist of $< f, M_f >$
$J(M_{f_i}, M_{f_j})$	Jaccard similarity between M_{f_i} and M_{f_j}
δ_J	Jaccard similarity threshold for strong co-occurrence, set to 0.5
δ_M	Prevalence threshold, set to 4
h	Random permutation function in MinHashing
b	Number of bands in locality-sensitive hashing
r	Number of MinHash values each band contains in locality-sensitive hashing
s	Jaccard similarity between a pair of files
TP	True positive; a malware instance correctly identified as bad
FP	False positive; a good file incorrectly identified as bad

Table 1: Main symbols used throughout the paper

3. PROPOSED METHOD: THE AESOP ALGORITHM

In this section, we describe the techniques that we used in designing AESOP so that it can scale to billions of files and over a hundred million machines. Figure 2 provides an overview of the AESOP approach. We begin by describing our use of MinHashing, which allows us to approximate the Jaccard similarity between two sets efficiently. Next, we explain our application of locality-sensitive hashing to efficiently identify peer-groups of strongly co-occurring files. Finally, we describe our use of belief propagation to propagate information from labeled files to their unlabeled peers.

3.1 MinHashing for Co-occurence Strength Estimation

It is not tractable to compute the Jaccard similarity between large sets due to the expensive set intersection and union operations it involves. *MinHashing* [5], which is short for *Minwise Independent Permutation Hashing*, is a popular technique to efficiently estimate the Jaccard similarity between two sets. MinHashing has been proven to work well for large scale real-world applications, such as detecting duplicate images [8] and clustering users on Google news [10]. We will explain MinHashing using dataset D in Table 2 as a running example. MinHashing *randomly* reorders the machines in M using a bijective function h that maps the machines in M to the set $\{1, ..., |M|\}$ in a random fashion. We call function h a *random permutation function*. An example function h for $M = \{m_1, ..., m_8\}$ is given in Table 2. Notice that if we rearrange the machines in $M_f \in D$ in ascending order of the machines' values retrieved from function h, we obtain a random permutation of M_f, which we refer to as M_f^h. For instance, M_{f_2} in Table 2 is permuted as $M_{f_2}^h = \{m_7, m_3, m_5\}$ since $h(m_7) = 1 < h(m_3) = 2 < h(m_5) = 8$. The MinHash value of M_f under function h, which we refer to as $h_{min}(M_f)$, is defined as $h_{min}(M_f) = \arg\min_{m_i \in M_f} h(m_i)$. Informally, $h_{min}(M_f)$ is the first element of M_f^h. For instance, $h_{min}(M_{f_1}) = m_4$ in Table 2.

File	Set of machines containing the file
f_1	$M_{f_1} = \{m_2, m_4, m_5, m_8\}$
f_2	$M_{f_2} = \{m_3, m_5, m_7\}$
f_3	$M_{f_3} = \{m_1, m_3, m_5, m_6, m_7\}$

$h(m_1) = 3$	$h(m_2) = 6$	$h(m_3) = 2$	$h(m_4) = 4$
$h(m_5) = 8$	$h(m_6) = 7$	$h(m_7) = 1$	$h(m_8) = 5$

Table 2: An example dataset D and random permutation function h

The key property of MinHashing is that the probability of the MinHash values of two sets being equal is equal to the Jaccard similarity between the sets. Formally, $\Pr(h_{min}(M_{f_i}) = h_{min}(M_{f_j})) = J(M_{f_i}, M_{f_j})$ (see Cohen et al. [9] or Rajaraman and Ullman [22] for a proof). As an example, in Table 2, $h_{min}(M_{f_1}) = m_4$, $h_{min}(M_{f_2}) = m_7$, and $J(M_{f_1}, M_{f_2}) = 0.17$.

3.2 Clustering Files

Despite the use of MinHashing, the number of file pairs to be considered for strong co-occurrence still remains very large for big datasets. It is also possible that two sets may not have the same MinHash value but in fact have a high Jaccard similarity, or may receive the same MinHash value but in fact have a low Jaccard similarity. Hence, a single MinHash value is typically not sufficient to deduce whether two sets have a high Jaccard similarity or, in our case, whether two files strongly co-occur. Locality-sensitive hashing (LSH), which we describe next, addresses these points.

LSH is a technique for approximate clustering and near-neighbor search in high dimensional spaces [16, 14]. Its main idea is to use multiple hash functions to map items into buckets such that similar items are more likely to be hashed to the same bucket. LSH uses *locality-sensitive function families* to achieve this goal.[2] At a high-level, each individual function in a locality-sensitive function family should be able to provide lower and upper bounds on the probability of whether two items with a pairwise similarity (or distance) in a particular interval will receive the same hash value from the function. Therefore, locality-sensitive function families are defined for particular similarity or distance measures, such as Hamming distance [14], L_p norms [14, 11], and earth mover's distance [6]. The random permutation functions used in MinHashing (see Section 3.1) form a locality-sensitive function family for the Jaccard similarity measure [9].

A nice property of the locality-sensitive function families is that they can be amplified by combining values returned from multiple functions via logical *AND* and/or *OR* [22]. In our context, given dataset D, this means we can compute n MinHash values—using n different random permutation functions—for each $M_f \in D$. Subsequently, these n MinHash values can be combined in multiple ways. An effective and generic way is to partition n MinHash values into b bands, each consisting of r values, such that $n = b \times r$.

As an example, consider Table 3, which lists six MinHash values for M_{f_4}, M_{f_5}, and M_{f_6} obtained from six different

[2] A function family is a group of functions that share certain characteristics.

	h_{min}	M_{f_4}	M_{f_5}	M_{f_6}	Buckets
Band 1	h^1_{min}	m_1	m_1	m_1	$[f_4, f_5]$ $[f_6]$
	h^2_{min}	m_1	m_1	m_2	
Band 2	h^3_{min}	m_5	m_5	m_3	$[f_4, f_5]$ $[f_6]$
	h^4_{min}	m_8	m_8	m_4	
Band 3	h^5_{min}	m_1	m_7	m_7	$[f_4][f_5, f_6]$
	h^6_{min}	m_6	m_6	m_6	

Table 3: Hypothetical inputs and outputs for LSH. The inputs are MinHash values for each file. The outputs are buckets containing files. This LSH scheme uses three bands, each consisting of two MinHash values.

random permutation functions $h^1, ..., h^6$. These six MinHash values are partitioned into three bands, each consisting of two values. For instance, M_{f_4}'s MinHash values for Band 2 are $\{m_5, m_8\}$. If we use a cryptographic hash function, such as SHA-256, to assign files to buckets based on their MinHash values in a band, then the files will appear in the same bucket if all of their r MinHash values in that band are the same. For instance, in Band 2 of Table 3, files f_4 and f_5 appear in the same bucket because their MinHash values for Band 2, i.e., h^3_{min} and h^4_{min}, are both $\{m_5, m_8\}$, whereas file f_6 appears in a separate bucket because its MinHash values are $\{m_3, m_4\}$. In this scheme, the files have b chances of appearing in the same bucket. This type of amplification is called an *AND*-construction with r rows followed by an *OR*-construction with b bands. This is because files will hash to the same bucket at least once if *all* of their r MinHash values (*AND* operation) in *any* of the b bands are the same (*OR* operation).

Our goal with LSH is that files f_i and f_j will appear together in at least one bucket if they strongly co-occur. Based on the scheme described above, we can derive the probability that files f_i and f_j will appear in at least one bucket given their true Jaccard similarity, $J(M_{f_i}, M_{f_j}) = s$, by following the derivation steps in Rajaraman and Ullman [22]. From Section 3.1, we know that the probability that one MinHash value of M_{f_i} and M_{f_j} being equal is s. Therefore, the probability of r MinHash values of M_{f_i} and M_{f_j} being the same is s^r. Notice that s^r is the probability that files f_i and f_j will hash to the same bucket in a particular band. Therefore, the probability that files f_i and f_j will not hash to the same bucket in a particular band is $1 - s^r$. Then, the probability that files f_i and f_j will not hash to the same bucket in all of the b bands is $(1 - s^r)^b$. Finally, the probability that files f_i and f_j will hash to the same bucket in at least one of the b bands is $1 - (1 - s^r)^b$.

3.2.1 Tuning Parameters of LSH

Figure 3 illustrates the effect of b and r on $1-(1-s^r)^b$ for various values of b, r, and s. Notice that increasing r allows us to prune file pairs with low Jaccard similarity whereas increasing b allows us to retain file pairs with high Jaccard similarity. As an example, consider our Jaccard similarity threshold for strong co-occurrence, $\delta_J = 0.5$. From the figure, we see that the probability that a pair of files with $s = 0.5$ appearing in at least one bucket is 0.5 when $b = 1$ and $r = 1$. The same probability drops to almost 0 when $b = 1$ and $r = 10$. Now, consider $s = 0.9$ with which we would deem a pair of files as strongly co-occurring. With $b = 1$ and $r = 10$, we see from the figure that the probability that the files in the pair appearing in at least one bucket is less than 0.4. The same probability increases to almost 1 when $b = 10$ and $r = 10$.

Given a δ_J value, one approach to select b and r is to first determine an r value that assigns to file pairs with $s \leq \delta_J$ a very small probability of appearing in at least one bucket. Then, we can select a b value that assigns to file pairs with $s > \delta_J$ a high probability of appearing in at least one bucket. Due to the "S" shape of the function $1 - (1 - s^r)^b$ [22], it is not feasible to ensure that all the file pairs with $s > \delta_J$ have a probability close to 1 while forcing any file pair with $s \leq \delta_J$ have a probability close to 0, similar to the ideal unit step function. Another consideration is that while large b and r values are advantageous, it also means that one needs to compute a significant amount of MinHash values for LSH. Given $\delta_J = 0.5$, we set $b = 10$ and $r = 10$ since these values prune the majority of the weakly co-occurring file pairs with $s \leq 0.5$ while retaining most of the strongly co-occurring file pairs with $s > 0.5$.

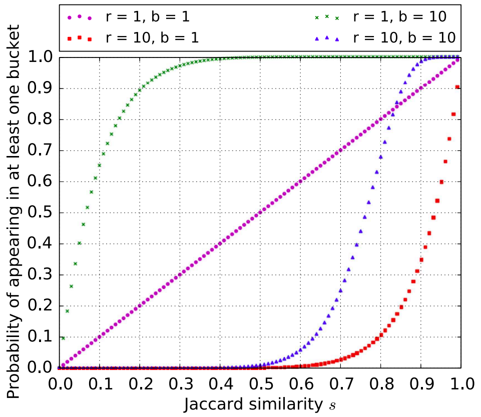

Figure 3: Effect of the number of bands, b, and MinHash values in each band, r, on the probability that two files with Jaccard similarity s will hash to the same bucket at least once in LSH. Given our Jaccard similarity threshold for strong co-occurrence $\delta_J = 0.5$, we set $b = 10$ and $r = 10$ (the blue curve with triangles) to prune the majority of the weakly co-occurring file pairs with $s \leq 0.5$ while retaining most of the strongly co-occurring file pairs with $s > 0.5$.

3.3 Labeling Files based on Co-occurrence

The outcome of LSH on dataset D is multiple bands, each consisting of a varying number of buckets that contain labeled and unlabeled files. A file appears at most once in a band, inside one of the buckets of the band. Notice that across different bands, the file might appear with a different set of files. For instance, in Table 3, file f_5 appears with file f_4 in two bands and with file f_6 in one band. In this section, we discuss how we combine the buckets from different bands into a unified "structure" and assign labels to unlabeled files using it.

Unipartite File Graph. Graphs provide a powerful representation of relationships between objects, therefore one approach to combine the buckets is to construct an undirected unipartite file graph by considering every pair of files in the buckets. In this graph, the files appear as nodes and they are connected with an edge if they *indeed* strongly co-occur. This additional check is performed to mitigate the effect of possible approximation errors incurred by LSH. The graph can then be used in a way that goodness and badness information is "sent" from the labeled files to the unlabeled files in the graph.

Our preliminary analyses showed that constructing a unipartite file graph as described above is not feasible. The reason is that a file inside a bucket typically strongly co-occurs with the majority of the other files in the bucket. This is most likely a property of the domain; there are intrinsic dependency relationships between files, e.g., the files under the "\Windows\System32" folder in the Windows operating systems. Therefore, large buckets contribute dense subgraphs to the graph during construction. In turn, the number of edges in the graph increases to billions, making it infeasible to operate on the graph.

Bipartite File-Bucket Graph. Due to this reason, Aesop instead operates on a undirected, unweighted bipartite file-bucket graph, which we also refer to as a *file-relation graph*. In this graph, there is an edge connecting a file node to a bucket node if the file appears in that bucket. Notice that the number of edges that would be included in the bipartite graph from a bucket of N strongly co-occurring files is $\mathcal{O}(N)$ in comparison to $\mathcal{O}(N^2)$ for the unipartite file graph. The bipartite graph is expected to contain more nodes than the unipartite file graph, however this is less of a concern for information propagation purposes as we will discuss. The bipartite graph is useful to assign labels to the unlabeled files; its difference with the unipartite file graph is that the files are now indirectly connected through the buckets, therefore goodness and badness information is first propagated from the labeled files to the buckets and then from the buckets to the unlabeled files.

Remarks. LSH ensures with high probability that if a file appears in a bucket consisting of more than one file, the file should strongly co-occur with at least one of the other files in the bucket. We represent this relationship by establishing an edge connecting the file to the bucket. It is, however, possible that a file does not strongly co-occur with some of the files in the bucket. In this case, the bipartite graph cannot capture the absence of strong co-occurrence between that set of files. In practice, this is a rare situation that does not pose any limitation on the effectiveness of Aesop, as we demonstrate in Section 4.

Benefits. A property of the bipartite graph is that it intrinsically captures the notion of a weight between the files. To illustrate this, consider two strongly co-occurring files f_i and f_j, and their Jaccard similarity $J(M_{f_i}, M_{f_j}) = s$. If we use a LSH scheme with b bands and r MinHash values as described in Section 3.2, the probability that files f_i and f_j appear in the same bucket in a band is s^r. Then, the number of bands files f_i and f_j appear together in a bucket is a random variable X that follows the Binomial distribution with parameters b and s^r, i.e., $X \sim \mathrm{B}(b, s^r)$. Thus, the larger the value of s, the more bands files in which f_i and f_j will appear together inside a bucket. In the bipartite graph, this results in a larger number of paths between files f_i and f_j that go through the buckets, thereby allowing files f_i and f_j to "influence" each other more than do the other files.

Pruning. After the bipartite graph is constructed, it is possible that some of its connected components consist of one file or only unlabeled files. These components do not contribute to solving the problem of assigning labels to unlabeled files, therefore Aesop discards them from the graph to retain only the useful information in the graph.

Belief Propagation. Next, we describe our approach to assign labels to unlabeled files using the bipartite graph. Our goal is to label the nodes corresponding to unlabeled files as *good* or *bad*, along with a measure of confidence. To achieve this, we treat each file as a random variable $X \in \{x_g, x_b\}$, where x_g is the *good* label and x_b is the *bad* label. The labels are simply the possible states for the random variable X. The file's goodness and badness can then be expressed by the probabilities $\Pr(x_g)$ and $\Pr(x_b)$, respectively, such that $\Pr(x_g) + \Pr(x_b) = 1$. Based on this formulation, our goal is to determine the marginal probabilities $\Pr(X_{f_i} = x_g)$ and $\Pr(X_{f_i} = x_b)$ for unlabeled file f_i. To achieve this, there exists important background information that we can leverage. First, we know that some nodes in the graph correspond to labeled files. Second, our intuition suggests homophilic relationships between the files, i.e., good files are expected to strongly co-occur with other good files and bad files are expected to strongly co-occur with other bad files.

The above formulation converts the bipartite graph into a pairwise Markov random field (MRF). The task of inferring the marginal distribution of each node in a pairwise MRF is NP-complete [25]. The Belief Propagation algorithm (BP) is a successful approximation technique for solving this problem. BP has been adapted to various domains, such as image restoration [13] and fraud detection [20]. The algorithm is also scalable; it takes time linear in the number of edges in the graph. For this reason, Aesop adapts BP to assign labels to unlabeled files.

At a high level, BP infers the marginal distribution of a node using some prior knowledge about the node and messages arriving from the node's neighbors. The idea is to iteratively pass messages between every pair of connected nodes i and j. Typically, $m_{ij}(x_k)$ represents the message sent from node i to node j, which denotes node i's belief that node j is in state x_k. The prior knowledge, or simply the *prior*, for node i is denoted by the node potential function ϕ_i that specifies the prior probabilities of node i being in the possible states. The message passing procedure stops when the messages converge or a maximum number of iterations is reached. The final, inferred marginal probabilities are called the final beliefs. The symbol $b_i(x_j)$ denotes the final belief that node i is in state x_j.

The BP algorithm is carried out as follows in practice. An edge between nodes i and j passes a message towards each direction for each possible state. The order of the transmission can be arbitrary if all the messages are passed in every iteration. The set of beliefs that a node has for each of its neighbors is kept normalized to sum to one. This prevents any numerical underflow, i.e., a certain belief reaching 0 due to limited precision. A message from node i to its neighbor node j is generated based on node i's neighbors' messages about node i. Formally, the message update equation is:

$$m_{ij}(x_k) \leftarrow \sum_{x_\ell \in X} \phi_i(x_\ell)\ \psi_{ij}(x_\ell, x_k)\ \frac{\prod_{p \in N(i)} m_{pi}(x_\ell)}{m_{ji}(x_\ell)}$$

where $N(i)$ is the set of nodes neighboring node i, and $\psi_{ij}(x_\ell, x_k)$ is called the *edge potential*; intuitively, it is a function that transforms a node's incoming messages into the node's outgoing messages. Formally, $\psi_{ij}(x_\ell, x_k)$ specifies the probability of node i being in state x_ℓ and node j being in state x_k.

Although BP is not theoretically guaranteed to converge in general graphs, in practice the algorithm usually converges quickly. After the message passing procedure stops and the algorithm ends, the final beliefs are computed as:

$$b_i(x_j) = k \ \phi_i(x_j) \prod_{p \in N(i)} m_{pi}(x_j)$$

where k is a normalizing constant.

AESOP's Adaptation of BP. Next, we describe how we map our background information into BP's context. Recall that there are two types of nodes in the bipartite graph: files and buckets. The nodes can be in either the good state or the bad state. For simplicity of exposition, we only mention the priors for the good state. For the buckets, we set their priors to 0.5. This is because we want a bucket to be initially neutral and influenced only by the files appearing in the bucket (thus connected to the bucket). For the labeled files, if the file is good we set its prior to 0.99 and if the file bad we set its prior to 0.01. For the unlabeled files, we set their priors to 0.5 so that they are initially neutral and their final beliefs are indirectly determined by the labeled files with which they strongly co-occur. We convert our intuition about homophilic file relationships into the edge potentials shown in Table 4, which indicate that a good (bad) file is more likely to be associated with a bucket consisting of other good (bad) files than a bucket consisting of bad (good) files.

$\psi_{ij}(x_\ell, x_k)$	$x_\ell = $ good	$x_\ell = $ bad
$x_k = $ good	0.99	0.01
$x_k = $ bad	0.01	0.99

Table 4: Edge potentials reflecting our intuition that it is more likely that good (bad) files strongly co-occur with other good (bad) files.

3.4 Time Complexity of AESOP

Aesop has two main components: LSH and BP. Given that the random permutation functions can be predetermined, LSH can be performed with a single scan of dataset D. At a high level, LSH considers each file in dataset D, maintaining a MinHash value for each permutation function while iterating over the set of machines it appears on. Assume that dataset D contains $|D|$ files. Also, recall that M denotes the set of all the machines. Then, |M| is the maximum number of machines a file can appear on. Hence, the time complexity for LSH is $\mathcal{O}(|D| \cdot |M|)$. The BP algorithm iterates over each edge in the file-relation graph a constant amount of times if the maximum number of iterations parameter is set. Assume that the graph has E edges. Then, the time complexity for BP is $\mathcal{O}(E)$. The overall time complexity for Aesop is therefore $\mathcal{O}(|D| \cdot |M| + E)$.

4. EXPERIMENTS

This section presents an experimental evaluation of Aesop. We measure its effectiveness at detecting labeled benign and malicious files as well as discovering labels for unlabeled benign and malicious files.

We conducted our experiments on a 64-bit Linux machine (RedHat Enterprise Linux Server 5.7) with 8 Opteron 2350 quad core processors running at 2.0 GHz, 64GB of RAM, and 100GB disk-quota per user.

4.1 Sampling Norton Community Watch

We leverage Symantec's Norton Community Watch data, the most important elements of which are unique file and machine identifiers. This terabyte-scale dataset contains more than 119 million machines and over 10.1 billion files. Due to the limited disk space budget, we obtained a sample of this dataset as follows.

Symantec's Worldwide Intelligence Network Environment (WINE) samples and aggregates datasets that Symantec uses in its day-to-day operations to share them with the research community [12]. The WINE sampling scheme selects machines uniformly at random and retrieves any data for the sampled machines from the production systems. Previous work showed that the uniform sampling of the machines is effective in terms of estimating or extrapolating crucial attributes of the original datasets from the samples [21].

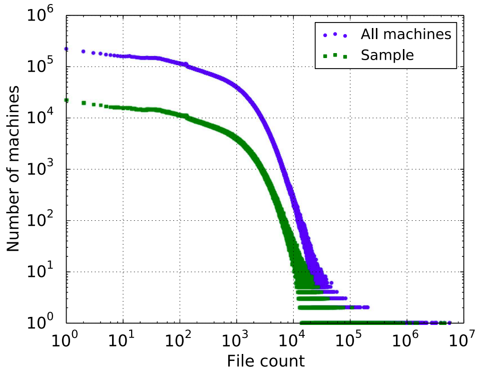

Figure 4: Distributions of the number of machines (vertical axis) with a particular file count (horizontal axis) for the full dataset (higher blue curve with circles) and the sample (lower green curve with rectangles). Our sampling strategy preserves the overall shape of the original distribution.

Motivated by this result, we employ a similar technique to sample machines from the Norton Community Watch dataset. The set of files appearing on each sampled machine is retrieved completely. Figure 4 shows the distributions of the number of machines containing a particular number of files for the original dataset and a 10% sample. More specifically, the number of machines in the sample is 10% of the total number of machines in the dataset. The uniform random sampling approach preserves the overall shape of the original distribution; both distributions are heavy-tailed with few machines containing a large number of files and a large number of machines containing few files.

We obtained the sample on November 6, 2013. After discarding the immature files with prevalence less than or equal to $\delta_M = 4$, the sample consists of $11,939,429$ machines and $43,353,581$ files, with labels for 7% of the files in the sample. The final sample dataset occupies 120GB space on disk.

4.2 AESOP File-relation Graph

From the sample, Aesop generates a file-relation graph of $6,056,802$ nodes and $19,103,825$ edges. The graph contains

1,663,506 good files, 47,956 bad files, and 1,085,937 unlabeled files, and 3,259,403 nodes that correspond to buckets. There are 40,556,182 files in the sample that do not appear in the graph because they appeared in connected components that consist of either only one file or only unlabeled files. These files are pruned from the graph for efficiency reasons as they provide no value. The number of buckets is large because Aesop uses 10 bands during LSH; each band contributes a similar set of files but a distinct set of buckets.

4.3 Sizes of Connected Components

Intuitively, Aesop's accuracy will be better if the files form small, disconnected clusters in the file-relation graph. This is because large groups of files are likely to contain a mix of good and bad files that are difficult to classify accurately. The connected components of a graph are its largest clusters, so in Figure 5 we show the graph's distribution of connected component sizes in terms of the number of files they contain. Note that the distribution is heavy tailed, indicating that most files appear in small-sized connected components. The graph's connected components that contain a very large number of files justify our selection of operating on a bipartite file-bucket graph instead of a unipartite file graph (see Section 3.3).

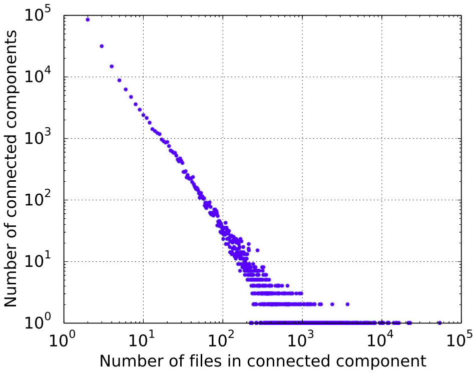

Figure 5: Distribution of the number of connected components (vertical axis) containing a particular number of files (horizontal axis) in Aesop's file-relation graph. Smaller components are less likely to contain a mix of good and bad files. The distribution is heavy tailed, indicating that most files appear in small-sized connected components.

4.4 Purity of Connected Components

It is also important that the file-relation graph's connected components are pure, i.e., they consist of files with identical labels. To test this, we turn to *entropy*, a widely used measure for determining the uncertainty or irregularity of a system [19]. We compute the entropy of a connected component as $(-\frac{e_g}{e_g+e_b}\log_2\frac{e_g}{e_g+e_b} - \frac{e_b}{e_g+e_b}\log_2\frac{e_b}{e_g+e_b})$, where e_g and e_b are the number of good and bad files in the component, respectively. Note that a smaller entropy denotes a purer connected component. Figure 6 shows the average entropy for the connected components containing a particular number of files. The error bars correspond to one standard deviation. We observe that a significant fraction of the connected components have entropies close to 0, indicating that they are pure regardless of their sizes.

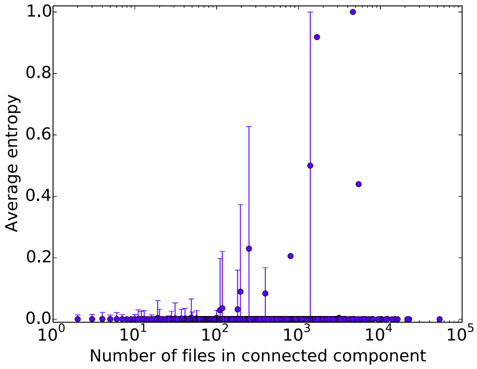

Figure 6: Average entropy for the connected components (vertical axis) containing a particular number of files (horizontal axis) in Aesop's file-relation graph. The error bars correspond to one standard deviation. A significant fraction of the connected components have 0 entropy, indicating that they consist of files with identical labels.

4.5 Performance Evaluation using Cross Validation

Next, we evaluate the effectiveness of Aesop in detecting benign and malicious files. Our evaluation scheme uses 10-fold cross validation. We treat the files in the test set as unlabeled files by setting their priors for the good state to 0.5. The files in the training set are assigned priors as described in Section 3.3. For each fold, we run the BP component of Aesop for 10 iterations and report the true positive (TP) rate at a fixed 0.0001 false positive (FP) rate. Recall that in our context a TP is a malware instance that is correctly identified as malicious and an FP is a benign file incorrectly identified as malicious.

The partitioning of the labeled files into disjoint sets may result in all of the labeled files in a connected component being assigned to the same set. When such a set is used as the test set, the corresponding connected component turns into a component consisting of only unlabeled files, which is undesirable, as it leaves behind no information to propagate in the component. For purposes of our evaluation, we prune any such connected component from the graph, as we do all the components that consist of only one file or only unlabeled files (see Section 3.3). This situation affects approximately 2% of the labeled files in the sample.

Figure 7 shows the overall and zoomed-in receiver operating characteristic (ROC) curves for this experiment. To obtain the ROC curve, we sort the final beliefs of all the files in ascending order and consider each value as a threshold; all files with final beliefs above that value are classified as good, or bad otherwise. Then, the TP rate and FP rate are computed using these classifications. We observe that Aesop achieves an impressive 0.9983 TP rate at the 0.0001 FP rate while labeling over 1.6 million files.

4.6 Early Discovery of Unlabeled Benign and Malicious Files

Next, we test the effectiveness of Aesop in assigning labels to unlabeled files. Recall that we obtained the sample on November 6, 2013. To determine to what extent Aesop labels files ahead of Symantec's current technology, which

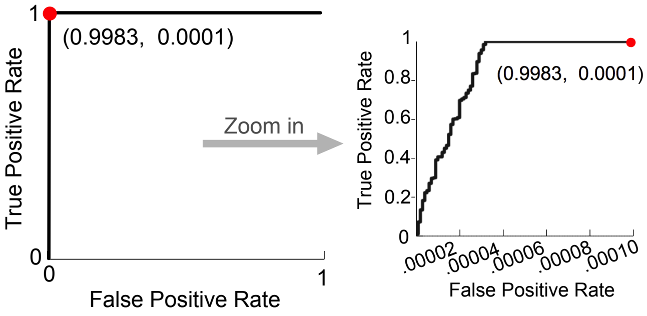

Figure 7: *Left*: ROC curve for the cross-validation experiment. Aesop achieves 0.9983 true positive rate at detecting malware, at 0.0001 false positive rate, while labeling over 1.6 million files. *Right*: Zoomed-in view.

includes the state-of-the-art Polonium approach [7], we also retrieved the file label information for November 13, 2013 and February 1, 2014. Here, we first use the labels from February 1 to obtain a list of files in Aesop's's file-relation graph that were unlabeled by Symantec on November 6, but are labeled as of February 1. There are 774 bad and 17,997 good such eventually-labeled files. For this experiment, we run the BP component of Aesop for 10 iterations using only the file label information available on November 6 with the eventually-labeled files as the test set.

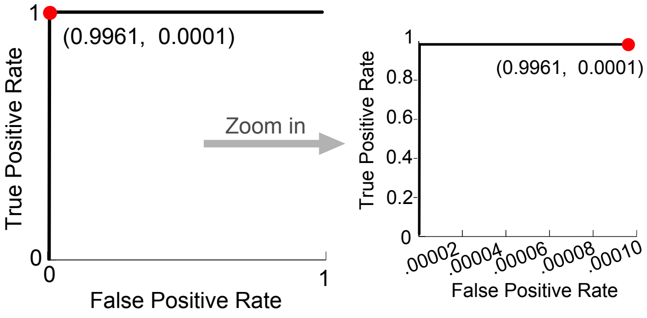

Figure 8: *Left*: ROC curve for the early discovery experiment. Aesop achieves 0.9961 true positive rate at detecting malware, at 0.0001 false positive rate, while labeling over 18 thousand originally unlabeled files. *Right*: Zoomed-in view.

Figure 8 shows the overall and zoomed-in receiver operating characteristic (ROC) curves for this experiment. We obtain the ROC curve as described in Section 4.5. We observe that Aesop achieves an impressive 0.9961 TP rate at the 0.0001 FP rate while labeling over 18 thousand originally unlabeled files.

To compare Aesop with the state-of-the-art Polonium approach [7], Figure 9 considers the file label information available on November 13 and shows the fractions of the eventually-labeled good and bad files that were still unlabeled on November 13. We observe that Aesop provides Symantec with at least a week's advantage in assigning labels to 99% and 79% of the eventually-labeled good and bad files, respectively, in comparison to Polonium.

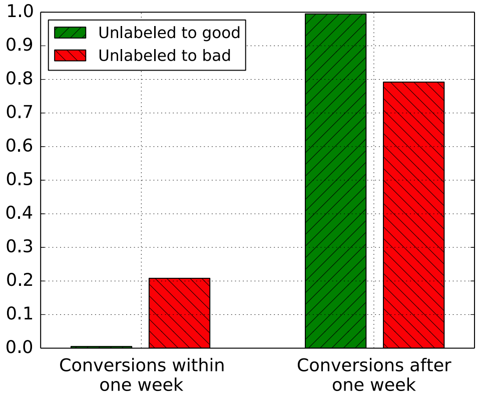

Figure 9: Fraction of unlabeled files that were and were not assigned labels within a week of the sample generation date. Aesop provides Symantec with at least a week's advantage in assigning labels to a significant amount of unlabeled files.

4.7 Running Times

Two main components of Aesop are LSH and BP. Figure 10 shows the average runtime of each component of Aesop on the sample over 3 executions, using 7 threads for LSH and a single thread for BP. The LSH step is fairly expensive, but it can be parallelized and scales linearly with the amount of input data. Also, recall that LSH processed 120GB data in this step. The BP step scales linearly with the number of edges in the file-relation graph. It can be scaled up even more through parallelization, e.g., using the MapReduce framework [17].

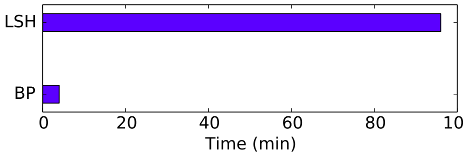

Figure 10: Average runtime for each component of Aesop on the sample occupying 120GB on disk.

5. TOWARDS DEPLOYMENT

We provide a brief discussion of the scope and limitations of our work, and then discuss our plans for integrating into Symantec's anti-malware solutions.

5.1 Scope and Limitations

It is important to describe two characteristics of this dataset that result in a significant fraction of immature files with weak co-occurrence that Aesop cannot label as benign or malicious. There are significant numbers of files that are either new or very rare, such as benign executable files created by developers that are never shared with other users, and malicious malware files that are entirely unique due to malware polymorphism. An additional limitation arises because the Norton Community Watch data provides only a partial view of the files that any given machine contains. Unfortunately, there are hundreds of thousands of machines for which our dataset only reports information about a single file (see Section 4.1). If a new or rare file only appears on a machine for which we have limited or no other information about co-occurring files, Aesop will consider it to be immature and will not be able to classify it.

Though the lack of complete machine-file data does limit Aesop's coverage, in the security space, this is a much better problem to have than that of labeling files incorrectly. In academic literature, a false positive rate of 0.1% is often considered to be good, however, this represents incorrectly labeling 1 out of a 1000 benign files as malicious. Since malicious file labels are ideally used to prevent malware infections by removing them from users' computers, this false positive rate is prohibitively high; the stability and usability of most customer machines would be impacted because of benign file removals. Ultimately, we do not expect our tool to be deployed in isolation of other techniques, but rather, that it will be an important component in Symantec's protection strategy and used to identify malicious files that cannot be identified through other techniques.

5.2 Deployment Plans

Our goal in this work was to determine how useful co-occurrence information is for malicious file detection, ultimately with an eye on integrating it into Symantec's suite of malware protection capabilities. Symantec's endpoint security products already incorporate many tiers of defence, including traditional Anti-Virus protection, Intrusion Detection, Behavioral Protection, and Insight^{TM}, of which Polonium's [7] machine-hygiene-based scores are a key component. We expect Aesop to slot easily into this existing suite of protection capabilities for the following reasons:

- Given Aesop's exceptionally low false positive rate, its malicious vs. benign verdicts are independently useful, providing improved coverage and early detection of malware samples.
- Symantec has an infrastructure for cloud-based querying of Insight scores that we can leverage for Aesop. This infrastructure is needed because Insight scores every file that Symantec has ever seen, which represents far more information than can be replicated on individual customer devices. Aesop produces a similar amount of data, but can leverage Insight's infrastructure.
- We developed Aesop on the same database that supports Insight, so that our reputation scores can be easily incorporated into Insight's reputation scores in the same way that Polonium's [7] file-reputation scores were used. That is, Aesop scores will form a new feature used by the Insight classifier, thereby improving its coverage and creating new opportunities for early malware detection.

The next step towards the deployment of Aesop is a more detailed study of its false positives, to determine what kinds of mistakes our approach is prone to making. We are also exploring additional applications for the reputation scores obtained from Aesop, such as for false-positive mitigation of existing technologies. In parallel, we believe that the file-relation dataset that this work creates will prove to be fertile ground for additional insights. For example, we expect to see clusters of files that contain many files from a single malware family. These clusters may offer an early view of emerging malware and insights into the attackers themselves.

6. RELATED WORK

The exceptional depth and breadth of related work in the malware-detection space is a testament to the importance and difficulty of the problem. Most closely related to Aesop's malware detection approach are reputation-based techniques and techniques that exploit similarities between files for detection.

There exist reputation systems that have been developed to address security-related problems, such as reputation scoring for IP addresses [1] and DNS entries [2, 3]. The most closely related work to ours is Polonium [7], one of Symantec's current malware detection technologies. Polonium also takes a graph-based approach to infer file reputation, however with important differences. First, Aesop infer files' goodness by directly considering file-to-file relations, which is different than Polonium's indirect approach of analyzing file-to-machine relations. Second, Polonium was not designed to pick out related files that frequently co-appear, while Aesop does; leveraging this relational information, Aesop is able to accurately label many files at least one week before the current technologies (as discussed in Section 4).

As the number of unique malware executable files has exploded due to their use of polymorphic and metamorphic techniques, security researchers are increasingly turning to techniques that identify clusters of related malware files rather than attempt to detect files individually. Symantec's MutantX-S [15] system clusters executables according to their static and dynamic properties. This approach works with low-level malware features such as sequences of machine-language opcodes, making it largely orthogonal to our approach.

Karampatziakis et al. [18] use file placement as the primary component of its malware detection technique, by leveraging unique properties of file containers that would not generalize to machines, such as the idea that the presence of any malicious file in an archive is sufficient evidence to label all files in that archive as malicious. In addition, rather than performing inference as Aesop does with belief propagation, their logistic regression classifier only looks at a file's immediate neighbors in the archives to which it belongs.

Ye et al. [24] presents a malware detection approach that combines file-to-file relationship data with features extracted at the individual file level. An important difference is the size of our dataset, which is orders of magnitude larger than their dataset of $691,650$ files. Second, due to the smaller dataset size, they do not address scalability challenges as we do. Third, while Aesop identifies malware at an extremely low false positive rate of 0.0001 by itself, their combined approach operates at a more than 0.001 false positive rate.

In summary, not only does Aesop demonstrate the independent value of calculating file-to-file similarity scores, it also provides an algorithm that addresses scalability problems while achieving impressive results compared to the state-of-the-art techniques. Furthermore, Aesop's belief propagation approach provides a reputation-based system with nuanced scores that are ideally suited for integrating and improving existing malware detection technologies.

7. CONCLUSIONS

We present Aesop, an algorithm that uses the principle of guilt by association to establish nuanced reputation scores for executable files based on the company they keep. We use a large dataset voluntarily contributed by the members of Norton Community Watch, consisting of partial lists of the files that exist on their machines. Aesop leverages locality-sensitive hashing to efficiently compute file similarity values to construct a file-relation graph for inferring file goodness based on belief propagation. Our experiments show that Aesop achieves earlier detection of unlabeled files with exceptionally low error rates in comparison to the state-of-the-art techniques.

8. REFERENCES

[1] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: Characterizing internet scam hosting infrastructure. In *Proceedings of the USENIX Security Symposium*, 2007.

[2] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for dns. In *Proceedings of the USENIX Security Symposium*, 2010.

[3] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the Annual Network and Distributed System Security Symposium*, 2011.

[4] Bleeping Computer. Cryptolocker ransomware information guide and faq. October 2013. `http://www.bleepingcomputer.com/virus-removal/cryptolocker-ransomware-information`.

[5] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, 1997.

[6] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the ACM Symposium on the Theory of Computing*, 2002.

[7] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the SIAM International Conference on Data Mining*, 2011.

[8] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. In *British Machine Vision Conference*, 2008.

[9] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. In *Proceedings of the IEEE International Conference on Data Engineering*, 2000.

[10] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the International conference on World Wide Web*, 2007.

[11] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the ACM Symposium on Computational Geometry*, 2004.

[12] T. Dumitras and D. Shou. Toward a standard benchmark for computer security research: The worldwide intelligence network environment (wine). In *Proceedings of the European Conference on Computer Systems BADGERS Workshop*, 2011.

[13] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision*, 70(1):41–54, 2006.

[14] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*, 1999.

[15] X. Hu, S. Bhatkar, K. Griffin, and K. G. Shin. Mutantx-s: Scalable malware clustering based on static features. In *Proceedings of USENIX Annual Technical Conference*, 2013.

[16] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the ACM Symposium on the Theory of Computing*, 1998.

[17] U. Kang, D. H. Chau, and C. Faloutsos. Mining large graphs: Algorithms, inference, and discoveries. In *Proceedings of the IEEE International Conference on Data Engineering*, 2011.

[18] N. Karampatziakis, J. W. Stokes, A. Thomas, and M. Marinescu. Using file relationships in malware classification. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment*, 2012.

[19] H. Kim and H. Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12):1495–1502, 2007.

[20] M. McGlohon, S. Bay, M. G. Anderle, D. M. Steier, and C. Faloutsos. Snare: A link analytic system for graph labeling and risk detection. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2009.

[21] E. E. Papalexakis, T. Dumitras, D. H. P. Chau, B. A. Prakash, and C. Faloutsos. Spatio-temporal mining of software adoption & penetration. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2013.

[22] A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2012.

[23] Symantec. Internet security threat report. 18, 2013.

[24] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu. Combining file content and file relations for cloud based malware detection. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, 2011.

[25] J. Yedidia, W. Freeman, and Y. Weiss. *Understanding belief propagation and its generalizations*, pages 239–270. Morgan Kaufmann Publishers Inc., 2003.