Polonium: Tera-Scale Graph Mining and Inference for Malware Detection

Duen Horng Chau
Carnegie Mellon University
dchau@cs.cmu.edu

Carey Nachenberg
Symantec
cnachenberg@symantec.com

Jeffrey Wilhelm
Symantec
jeffrey_wilhelm@symantec.com

Adam Wright
Symantec
adam_wright@symantec.com

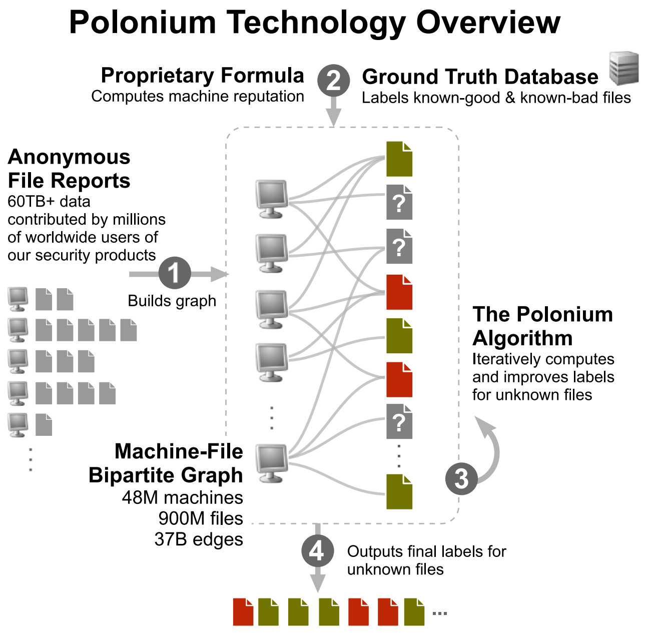
Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

Abstract

We present *Polonium*, a novel Symantec technology that detects malware through large-scale graph inference. Based on the scalable Belief Propagation algorithm, Polonium infers every file's reputation, flagging files with *low reputation* as *malware*. We evaluated Polonium with a billion-node graph constructed from the largest file submissions dataset ever published (60 terabytes). Polonium attained a high *true positive rate* of 87% in detecting malware; in the field, Polonium lifted the detection rate of existing methods by 10 *absolute* percentage points. We detail Polonium's design and implementation features instrumental to its success. Polonium has served 120 million people and helped answer more than *one trillion* queries for file reputation.

1 Introduction and Motivation.

Thanks to ready availability of computers and ubiquitous access to high-speed Internet connections, malware has been rapidly gaining prevalence over the past decade, spreading and infecting computers around the world at an unprecedented rate. In 2008, Symantec, a global security software provider, reported that the release rate of malicious code and other unwanted programs may be exceeding that of legitimate software applications [1]. This suggests traditional signature-based malware detection solutions will face great challenges in the years to come, as they will likely be outpaced by the threats created by malware authors. To put this into perspective, Symantec reported that they released nearly 1.8 million virus signatures in 2008, resulting in 200 million detections per month in the field [1]. While this is a large number of blocked malware, a great deal more malware (so-called "zero day" malware [2]) is being generated or mutated for each victim or small number of victims, which tends to evade



Polonium Technology Overview
Proprietary Formula
Computes machine reputation
2
Ground Truth Database
Labels known-good & known-bad files
Anonymous File Reports
60TB+ data contributed by millions of worldwide users of our security products
1
Builds graph
The Polonium Algorithm
Iteratively computes and improves labels for unknown files
Machine-File Bipartite Graph
48M machines
900M files
37B edges
3
4
Outputs final labels for unknown files

Figure 1: Overview of the Polonium technology

traditional signature-based antivirus scanners. This has prompted the software security industry to rethink their approaches in detecting malware, which have heavily relied on refining existing signature-based protection models pioneered by the industry decades ago. A new, radical approach to the problem is needed.

The New Polonium Technology. Symantec introduced a protection model that computes a *reputation* score for every application that users may encounter, and protects them from those with *poor reputation*. Good applications typically are used by many users, from known publishers, and have other attributes that characterize their legitimacy and good reputation. Bad applications, on the other hand, typically come from

Technical term	Synonyms	Meaning
Malware	Bad software, malicious software, infected file	Short for malicious software, which includes computer viruses, Trojan, etc.
Reputation	Goodness, belief (when discussing the Polonium algorithm)	A measure of the goodness; can be used on *machines* and *files* (e.g., file reputation)
File	Executable, software, application, program	A software instance, typically an executable (e.g., .exe) on the user's computer
Machine	Computer	A user's computer; a user can have multiple computers
File ground truth	–	File label, *good* or *bad*, assigned by human security experts
Known-good file	–	File with *good* ground truth
Known-bad file	–	File with *bad* ground truth
Unknown file	–	File with *unknown* ground truth
Positive (as in *true positive*)	–	Malware instance
True Positive	TP	Malware instance correctly identified as bad
False Positive	FP	A good file incorrectly identified as bad

Table 1: Malware detection terminology

unknown publishers, have appeared on few computers, and have other attributes that indicate poor reputation. The application reputation is computed by leveraging tens of terabytes of data anonymously contributed by millions of volunteers using Symantec's security software. These data contain important characteristics of the applications running on their systems.

In this paper, we describe Polonium, a new malware detection technology developed at Symantec that computes application reputation (Figure 1). We designed Polonium to complement (*not* to replace) existing malware detection technologies to better protect computer users from security threats. Polonium stands for "**P**ropagation **Of** **L**everage **Of** **N**etwork **I**nfluence **U**nearths **M**alware". Our main contributions are:

- Formulating the classic malware detection problem as a large-scale graph mining and inference problem, where the goals are to infer the reputation of any files that computer users may encounter, and identify the ones with poor reputation (i.e., malware). [Section 4]
- Providing an algorithm that efficiently computes application reputation. In addition, we show how domain knowledge is readily incorporated into the algorithm to identify malware. [Section 4]
- Investigating patterns and characteristics observed in a large anonymized file submissions dataset (*60 terabytes*), and the machine-file bipartite graph constructed from it (*37 billion* edges). [Section 3]
- Performing a large-scale evaluation of Polonium over a real, billion-node machine-file graph, demonstrating that our method is fast, effective, and scalable. [Section 5]
- Evaluating Polonium in the field, while it is serving 120 million users worldwide. Security experts investigated Polonium's effectiveness and found that it helped significantly lift the detection rate of a collection of existing proprietary methods by more than 10 *absolute* percentage points. To date, Polonium has helped answer more than *one trillion* queries for file reputation. [Section 6]

To enhance readability of this paper, we have listed the malware detection terminology used in this paper in Table 1. The reader may want to return to this table throughout this paper for technical terms' meanings and synonyms used in various contexts of discussion. One important note is that we will use the words "file", "application", and "executable" interchangeably to refer to any piece of software running on a user's computer, whose legitimacy (*good* or *bad*) we would like to determine.

2 Background and Our Differences.

To the best of our knowledge, formulating the malware detection problem as a file reputation inference problem over a machine-file bipartite graph is novel. Our work intersects the domains of malware detection and graph mining, and we briefly review related work below.

A *malware instance* is a program that has malicious intent [3]. *Malware* is a general term, often used to describe a wide variety of malicious code, including viruses, worms, Trojan horses, rootkits, spyware, adware, and more [4]. While some types of malware, such as viruses, are certainly malicious, some are on the borderline. For example, some "less harmful" spyware programs collect the user's browsing history, while the "more harmful" ones steal sensitive information such as credit card numbers and passwords; depending on what it collects, a spyware can be considered malicious, or only undesirable.

The focus of our work is *not* on classifying software into these, sometimes subtle, malware subcategories. Rather, our goal is to come up with a new, high-level method that can automatically identify more malware instances similar to the ones that have already been flagged by our company as harmful and that the user should remove immediately, or would be removed automatically for them by our security products. This distinction differentiates our work from existing ones that target specific malware subcategories.

2.1 Research in Malware Detection. There has been significant research in most malware categories. Idika and Mathur [5] comprehensively surveyed 45 state-of-the-art malware detection techniques and broadly divide them into two categories: (1) *anomaly-based detection*, which detects malware's deviation from some presumed "normal" behavior, and (2) *signature-based detection*, which detects malware that fits certain profiles (or signatures).

There have been an increasing number of researchers who use data mining and machine learning techniques to detect malware [6]. Kephart and Arnold [7] were the pioneers in using data mining techniques to automatically extract virus signatures. Schultz et al. [8] were among the first who used machine learning algorithms (Naive Bayes and Multi-Naive Bayes) to classify malware. Tesauro et al. [9] used Neural Network to detect "boot sector viruses", with over 90% true positive rate in identifying those viruses, at 15-20% false positive rate; they had access to fewer than 200 malware samples. One of the most recent work by Kolter and Maloof [10] used TFIDF, SVM and decision trees on n-grams.

Most existing research only considers the intrinsic characteristics of the malware in question, but has not taken into account those of the machines that have the malware. Our work makes explicit our strong leverage in propagating and aggregating machine reputation information for a file to infer its goodness.

Another important distinction is the size of our real dataset. Most earlier works trained and tested their algorithms on file samples in the thousands; we have access to over 900M files, which allows us to perform testing in a much larger scale.

2.2 Research in Graph Mining. There has been extensive work done in graph mining, from authority propagation to fraud detection, which we will briefly review below.

Authority & Trust Propagation: Finding authoritative nodes is the focus of the well-known PageRank [11] and HITS [12] algorithms; at the high level, they both consider a webpage as "important" if other "important" pages point to it. In effect, the importance of webpages are propagated over hyperlinks connecting the pages. TrustRank [13] propagates trust over a network of webpages to identify useful webpages from spam (e.g., phishing sites, adult sites, etc.). Tong et al. [14] uses *Random Walk with Restart* to find arbitrary user-defined subgraphs in an attributed graph. For the case of propagation of two or more competing labels on a graph, semi-supervised learning methods [15] have been used. Also related is the work on relational learning by Neville et al. [16, 17], which aggregates features across nodes to classify movies and stocks.

Fraud Detection & Graph Mining : Graph mining methods have been successfully applied in many domains. However, less graph mining research is done in the malware detection domain. Recent works, such as [3, 18], focus on detecting malware variants through the analysis of *control-flow graphs* of applications.

Fraud detection is a closely related domain. The NetProbe system [19] models eBay users as a tripartite graph of *honest* users, *fraudsters*, and their *accomplices*; NetProbe uses the Belief Propagation algorithm to identify the subgraphs of fraudsters and accomplices lurking in the full graph. McGlohon et al. [20] proposed the general SNARE framework based on standard Belief Propagation [21] for general labeling tasks; they demonstrated the framework's success in pinpointing misstated accounts in some general ledger data.

More generally, [22, 23] use knowledge about the social network structure to make inference about the key agents in networks. There is also a wealth of algorithms for mining frequent subgraphs such as *gSpan*[24], the GraphMiner system [25] and related systems [26, 27, 28].

3 Data Description.

Now, we describe the large dataset that the Polonium technology leverages for inferring file reputation.

Source of Data: Since 2007, tens of millions of worldwide users of Symantec's security products volunteered to submit their application usage information to us, contributing anonymously to help with our effort in computing file reputation. At the end of September 2010, the total amount of raw submission data has reached

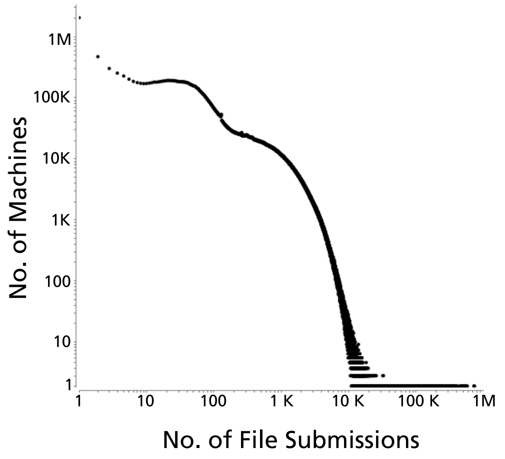


Figure 2: Machine submission distribution (log-log)

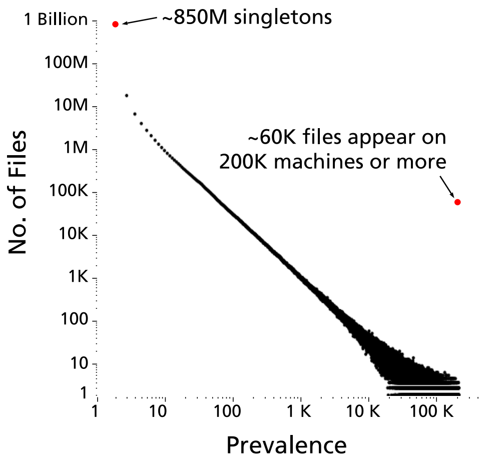


Figure 3: File prevalence distribution, in log-log scale. Prevalence cuts off at 200,000 which is the maximum number of machine associations stored for each file. Singletons are files reported by only one machine.

110 terabytes. We use a 3-year subset of these data, from 2007 to early 2010, to describe our method (Section 4) and to evaluate it (Section 5).

These raw data are anonymized; we have no access to personally identifiable information. They span over *60 terabytes* of disk space. We collect statistics on both legitimate and malicious applications running on each participant's machine — this application usage data serves as input to the Polonium system. The total number of unique files described in the raw data exceeds 900M. These files are executables (e.g., exe, dll), and throughout this paper, we will simply call them "files".

After our teams of engineers collected and processed these raw data, we constructed a huge bipartite graph from them, with almost *one billion* nodes and *37 billion* edges. To the best of our knowledge, both the raw file submission dataset and this graph are the largest of their kind ever published. We note, however, these data are only from a subset of our company's complete user base.

Each contributing machine is identified by an anonymized *machine ID*, and each file by a *file ID* which is generated based on a cryptographically-secure hashing function.

Machine & File Statistics: A total of 47,840,574 machines have submitted data about files on them. Figure 2 shows the distributions of the machines' numbers of submissions. The two modes approximately correspond to data submitted by two major versions of our security products, whose data collection mechanisms differ. Data points on the left generally represent new machines that have not submitted many file reports yet; with time, these points (machines) gradually move towards the right to join the dominant distribution.

903,389,196 files have been reported in the dataset. Figure 3 shows the distribution of the file prevalence, which follows the Power Law. As shown in the plot, there are about 850M files that have only been reported once. We call these files "singletons". We believes that these singleton files fall into two different categories:

- Malware which has been mutated prior to distribution to a victim, generating a unique variant;

- Legitimate software applications which have their internal contents fixed up or JITted during installation or at the time of first launch. For example, Microsoft's .NET programs are JITted by the .NET runtime to optimize performance; this JITting process can result in different versions of a baseline executable being generated on different machines.

For the files that are highly prevalent, we store only the first 200,000 machine IDs associated with those files.

Bipartite Graph of Machines & Files: We generated an undirected, unweighted bipartite machine-file graph from the raw data, with almost 1 billion nodes and 37 billion edges (37,378,365,220). 48 million of the nodes are machine nodes, and 903 million are file nodes. An (undirected) edge connects a file to a machine that has the file. All edges are unweighted; at most one edge connects a file and a machine. The graph is stored on disk as a binary file using the *adjacency list* format, which spans over 200GB.

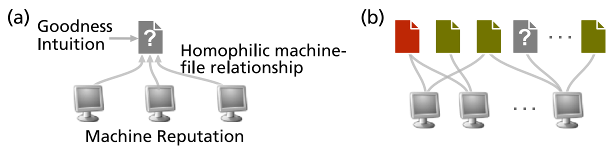


Figure 4: Inferring file goodness through incorporating (a) domain knowledge and intuition, and (b) other files' goodness through their influence on associated machines.

4 Proposed Method: the Polonium Algorithm.

In this section, we present the Polonium algorithm for detecting malware. We begin by describing the malware detection problem and enumerating the pieces of helpful domain knowledge and intuition for solving the problem.

4.1 Problem Description.

Our Data: We have a billion-node graph of machines and files, and we want to label the files node as *good* or *bad*, along with a measure of confidence in those dispositions. We may treat each file as a random variable $X \in \{x_g, x_b\}$, where x_g is the *good* label (or class) and x_b is the *bad* label. The file's goodness and badness can then be expressed by the two probabilities $P(x_g)$ and $P(x_b)$ respectively, which sum to 1.

Goal: We want to find the marginal probability $P(X_i = x_g)$, or goodness, for each file i. Note that as $P(x_g)$ and $P(x_b)$ sum up to one, knowing the value of one automatically tells us the other.

4.2 Domain Knowledge & Intuition.

For each file, we have the following pieces of domain knowledge and intuition, and we would like to use them to help infer the file's goodness, as depicted in Figure 4a.

Machine Reputation: A reputation score has been computed for each machine based on a proprietary formula that takes into account multiple anonymous aspects of the machine's usage and behavior. The score is a value between 0 and 1. Intuitively, we expect files associated with a good machine to be more likely to be good.

File Goodness Intuition: Good files typically appear on many machines and bad files appear on few machines.

Homophilic Machine-File Relationships. We expect that good files are more likely to appear on machines with good reputation and bad files more likely to appear on machines with low reputation. In other words, the machine-file relationships can be assumed to follow homophily.

File Ground Truth: We maintain a *ground truth database* that contains large number of *known-good* and *known-bad* files, some of which exist in our graph. We can leverage the labels of these files to infer those of the unknowns. The ground truth files influence their associated machines which indirectly transfer that influence to the unknown files. This intent is depicted in Figure 4b.

The attributes mentioned above are just a small subset of the vast number of machine- and file-based attributes we have analyzed and leveraged to protect users from security threats.

4.3 Formal Problem Definition

After explaining our goal and information we are equipped with to detect malware, now we formally state the problem as follows.
Given:

- An undirected graph $G = (V, E)$ where the nodes V correspond to the collection of files and machines in the graph, and the edges E correspond to the associations among the nodes.
- Binary class labels $X \in \{x_g, x_b\}$ defined over V
- Domain knowledge that helps infer class labels

Output: Marginal probability $P(X_i = x_g)$, or goodness, for each file.

Our goal task of computing the goodness for each file over the billion-node machine-file graph is an NP-hard inference task [21]. Fortunately, the Belief Propagation algorithm (BP) has been proven very successful in solving inference problems over graphs in various domains (e.g., image restoration, error-correcting code). We adapted the algorithm for our problem, which was a non-trivial process, as various components used in the algorithm had to be fine tuned; more importantly, as we shall explain, modification to the algorithm was needed to induce iterative improvement in file classification.

At the high level, the algorithm infers the label of a node from some prior knowledge about the node, and from the node's neighbors. This is done through iterative message passing between all pairs of nodes v_i and v_j. Let $m_{ij}(x_j)$ denote the message sent from i to j. Intuitively, this message represents i's opinion about j's likelihood of being in class x_j. The prior knowledge about a node i, or the prior probabilities of the node being in each possible class are expressed through the *node potential function* $\phi(x_i)$, which we shall discuss shortly. This prior probability is also called a *prior*.

At the end of the procedure, each file's goodness is determined. This goodness is an estimated marginal probability, and is also called *belief*, or formally $b_i(x_i)$ ($\approx P(x_i)$), which we can threshold into one of the binary classes. For example, using a threshold of 0.5, if the file belief falls below 0.5, the file is considered bad.

In details, messages are obtained as follows. Each edge e_{ij} is associated with messages $m_{ij}(x_j)$ and $m_{ji}(x_i)$ for each possible class. Provided that all messages are passed in every iteration, the order of passing can be arbitrary. Each message vector m_{ij} is normalized over j (node j is the message's recipient), so that it sums to one. Normalization also prevents numerical underflow (or zeroing-out values). Each outgoing message from a node i to a neighbor j is generated based on the incoming messages from the node's other neighbors. Mathematically, the message-update equation is:

$$m_{ij}\left(x_j\right) \leftarrow \sum_{x_i \in X} \phi\left(x_i\right) \psi_{ij}\left(x_i, x_j\right) \prod_{k \in N(i) \backslash j} m_{ki}\left(x_i\right)$$

where $N\left(i\right)$ is the set of nodes neighboring node i, and $\psi_{ij}\left(x_i, x_j\right)$ is called the *edge potential*; intuitively, it is a function that *transforms* a node's incoming messages into the node's outgoing ones. Formally, $\psi_{ij}\left(x_i, x_j\right)$ equals the probability of a node i being in class x_i given that its neighbor j is in class x_j. We shall explain how this function is tailored to our problem.

The algorithm stops when the beliefs converge (within some threshold; 10^{-5} is commonly used), or a maximum number of iterations has finished. Although convergence is not guaranteed theoretically for general graphs (except for trees), the algorithm often converges quickly in practice. When the algorithm ends, the node beliefs are determined as follows:

$$b_i\left(x_i\right) = k\phi\left(x_i\right) \prod_{x_j \in N(i)} m_{ji}\left(x_i\right)$$

where k is a normalizing constant.

4.4 The Polonium Adaptation of Belief Propagation (BP).

Now, we explain how we solve the challenges of incorporating domain knowledge and intuition to achieve our goal of detecting malware. Succinctly, we can map our domain knowledge and intuition to BP's components (or functions) as follows.

Machine-File Relationships → Edge Potential
We convert our intuition about the machine-file homophilic relationship into the *edge potential* shown in Figure 5, which indicates that a good file is slightly more likely to be associated with a machine with good reputation than with a low-reputation one. (Similarly for bad file.) ϵ is a small value (we chose 0.001), so that the fine differences between probabilities can be preserved.

$\psi_{ij}\left(x_i, x_j\right)$	$x_i = \text{good}$	$x_i = \text{bad}$
$x_j = \text{good}$	$0.5 + \epsilon$	$0.5 - \epsilon$
$x_j = \text{bad}$	$0.5 - \epsilon$	$0.5 + \epsilon$

Figure 5: Edge potentials indicating homophilic machine-file relationship. We choose $\epsilon = 0.001$ to preserve minute probability differences

Machine Reputation → Machine Prior
The *node potential function* for machine nodes maps each machine's *reputation score* into the machine's *prior*, using an exponential mapping (see Figure 6a) of the form

$$machine\ prior = e^{-k \times reputation}$$

where k is a numerical constant internally determined based on domain knowledge.

File Goodness Intuition → Unknown-File Prior
Similarly, we use another *node potential function* to set the file *prior* by mapping the intuition that files that have appeared on many machines (i.e., files with high prevalence) are typically good. Figure 6b shows such a mapping.

File Ground Truth → Known-File Prior
For known-good files, we set their priors to 0.99. For known-bad, we use 0.01.

4.5 Modifying the File-to-Machine Propagation.

In standard Belief Propagation, messages are passed along both directions of an edge. That is, an edge is associated with a *machine→file* message, and a *file→machine* message.

We explained in Section 4 that we use the homophilic edge potential (see Figure 5) to propagate machine reputations to a file from its associated machines.

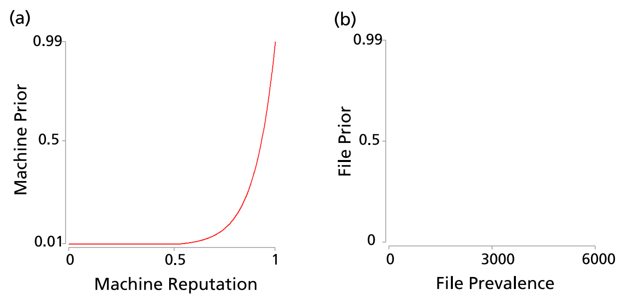


Figure 6: (a) Machine Node Potential (b) File Node Potential

Theoretically, we could also use the same edge potential function for propagating file reputation to machines. However, as we tried through numerous experiments — varying the ϵ parameter, or even "breaking" the homophily assumption — we found that machines' intermediate beliefs were often forced to changed too significantly, which led to an undesirable chain reaction that changes the file beliefs dramatically as well, when these machine beliefs were propagated back to the files. We hypothesized that this is because a machine's reputation (used in computing the machine node's prior) is a reliable indicator of machine's beliefs, while the reputations of the files that the machine is associated with are weaker indicators. Following this hypothesis, instead of propagating file reputation directly to a machine, we pass it to the formula used to generate machine reputation, which re-compute a new reputation score for the machine. Through experiments discussed in Section 5, we show that this modification leads to iterative improvement of file classification accuracy.

In summary, the key idea of the Polonium algorithm is that it infers a file's goodness by looking at its associated machines' reputations iteratively. It uses all files' current goodness to adjust the reputation of machines associated with those files; this adjusted machine reputation, in turn, is used for re-inferring the files' goodness.

5 Empirical Evaluation.

In this section, we show that the Polonium algorithm is scalable and effective at iteratively improving accuracy in detecting malware. We evaluated the algorithm with the bipartite machine-file graph constructed from the raw file submissions data collected during a three year period, from 2007 to early 2010 (as described in Section 3). The graph consists of about 48 million machine nodes and 903 million file nodes. There are 37 billion edges among them, creating the largest network of its type ever constructed or analyzed to date.

All experiments that we report here were run on a 64Bit Linux machine (Red Hat Enterprise Linux Server 5.3) with 4 Opteron 8378 Quad Core Processors (16 cores at 2.4 GHz), 256GB of RAM, 1 TB of local storage, and 60+ TB of networked storage.

One-tenth of the ground truth files were used for evaluation, and the rest were used for setting file priors (as "training" data). All TPRs (true positive rates) reported here were measured at 1% FPR (false positive rate), a level deemed acceptable for our evaluation. Symantec uses myriads of malware detection technologies; false positives from Polonium can be rectified by those technologies, eliminating most, if not all, of them. Thus, the 1% FPR used here only refers to that of Polonium, and is independent of other technologies.

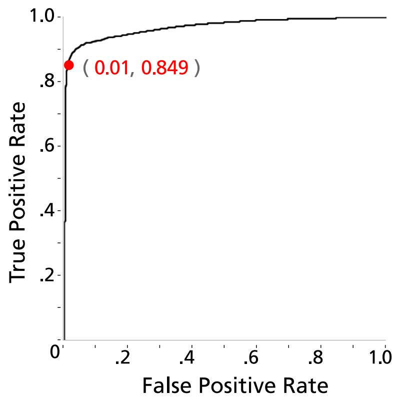


Figure 7: True positive rate and false positive rate for files with prevalence 4 and above.

5.1 Single-Iteration Results.

With one iteration, the algorithm attains 84.9% TPR, for all files with prevalence 4 or above[1], as shown in Figure 7. To create the smooth ROC curve in the figure, we generated 10,000 threshold points equidistant in the range $[0, 1]$ and applied them on the beliefs of the files in the evaluation set, such that for each threshold value, all files with beliefs above that value are classified as good, or bad otherwise. This process generates 10,000 pairs of TPR-FPR values; plotting and connecting these points gives us the smooth ROC curve as shown in Figure 7.

We evaluated on files whose prevalence is 4 or above. For files with prevalence 2 or 3, the TPR was only 48% (at 1% FPR), too low to be usable in practice. For completeness, the overall TPR for all files with prevalence 2 and higher is 77.1%. It is not unexpected, however, that the algorithm does not perform as effectively for low-prevalence files, because a low-prevalence file is associated with few machines. Mildly inaccurate information from these machines can affect the low-prevalence file's reputation significantly more so than that of a high-prevalence one. We intend to combine this technology with other complementary ones to tackle files in the full spectrum of prevalence.

5.2 Multi-Iteration Results.

The Polonium algorithm is iterative. After the first iteration, which attained a TPR of 84.9%, we saw a further improvement of about 2.2% over the next six iterations (see Figure 8), averaging at 0.37% improvement per itera-

[1] As discussed in Section 3, a file's prevalence is the number of machines that have reported it. (e.g., a file of prevalence five means it was reported by five machines.)

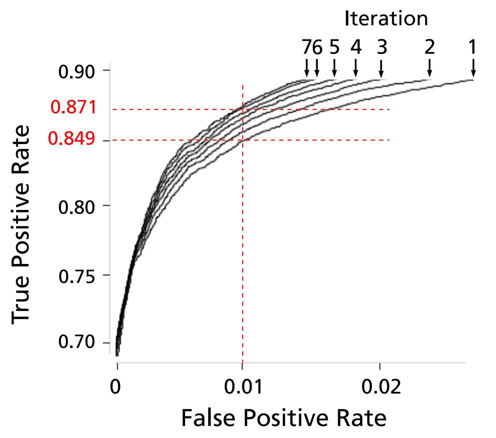


Figure 8: ROC curves of 7 iterations; true positive rate incrementally improves.

tion, where initial iterations' improvements are generally more than the later ones, indicating a diminishing return phenomenon. Since the baseline TPR at the first iteration is already high, these subsequent improvements represent some encouraging results.

5.2.1 Iterative Improvements. In Table 9, the first row shows the TPRs from iteration 1 to 7, for files with prevalence 4 or higher. The corresponding (zoomed-in) changes in the ROC curves over iterations is shown in Figure 8.

	Iteration							
Prev.	1	2	3	4	5	6	7	%↑
≥ 4	***84.9***	*85.5*	*86.0*	*86.3*	*86.7*	*86.9*	*87.1*	2.2
≥ 8	88.3	88.8	89.1	89.5	89.8	90.0	90.1	1.8
≥ 16	91.3	91.7	92.1	92.3	92.4	92.6	92.8	1.5
≥ 32	92.1	92.9	93.3	93.5	93.7	93.9	93.9	1.8
≥ 64	90.1	90.9	91.3	91.6	91.9	92.1	92.3	2.2
≥ 128	90.4	90.7	91.4	91.6	91.7	91.8	91.9	1.5
≥ 256	89.8	90.7	91.1	91.6	92.0	92.5	92.5	2.7

Figure 9: True positive rate (TPR, in %) in detecting malware incrementally improves over 7 iterations, across the file prevalence spectrum. Each row in the table corresponds to a range of file prevalence shown in the leftmost column (e.g., ≥ 4, ≥ 8). The rightmost column shows the absolute TPR improvement after 7 iterations.

We hypothesized that this improvement is limited to very-low-prevalence files (e.g., 20 or below), as we believed their reputations would be more easily influenced by incoming propagation than high-prevalence files. To verify this hypothesis, we gradually excluded the low-prevalence files, starting with the lowest ones, and observed changes in TPR. As shown in Table 9, even after excluding all files below 32 prevalence, 64, 128 and 256, we still saw improvements of more than 1.5% over 6 iterations, disproving our hypothesis. This indicate, to our surprise, that the improvements happen across the prevalence spectrum.

To further verify this, we computed the *eigenvector centrality* of the files, a well-known centrality measure defined as the principal eigenvector of a graph's adjacency matrix. It describes the "importance" of a node; a node with high eigenvector centrality is considered important, and it would be connected to other nodes that are also important. Many other popular measures, e.g., PageRank [11], are its variants. Figure 10 plots the *file reputation* scores (computed by Polonium) and the *eigenvector centrality* scores of the files in the evaluation set. Each point in the figure represents a file. We have zoomed in to the lower end of the centrality axis (vertical axis); the upper end (not shown) only consists of good files with reputations close to 1.

At the plot's upper portion, high centrality scores have been assigned to many good files, and at the lower portion, low scores are simultaneously assigned to many good and bad files. This tells us two things: (1) Polonium can classify most good files and bad files, whether they are "important" (high centrality), or less so (low centrality); (2) eigenvector centrality alone is unsuitable for spotting bad files (which have similar scores as many good files), as it only considers nodal "importance" but does not use the notion of *good* and *bad* like Polonium does.

5.2.2 Goal-Oriented Termination. An important improvement of the Polonium algorithm over Belief Propagation is that it uses a *goal-oriented* termination criterion—the algorithm stops when the TPR no longer

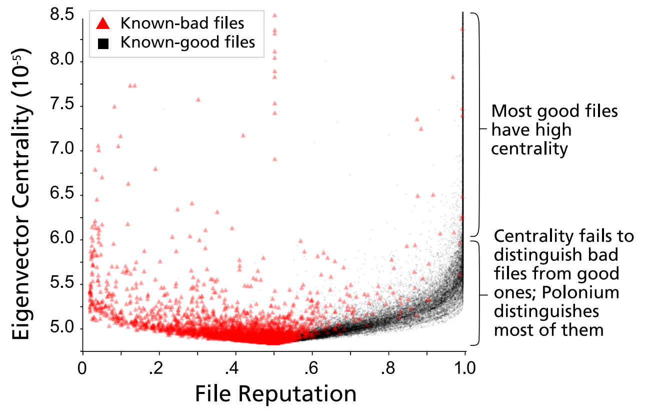


Figure 10: *File reputation* scores versus *eigenvector centrality* scores for files in the evaluation set.

increases (at the preset 1% FPR). This is in contrast to Belief Propagation's conventional *convergence-oriented* termination criterion. In our premise of detecting malware, the goal-oriented approach is more desirable, because our goal is to classify software into good or bad, at as high of a TPR as possible while maintaining low FPR — the convergence-oriented approach does not promise this; in fact, node beliefs can converge, but to undesirable values that incur poor classification accuracy. We note that in each iteration, we are trading FPR for TPR. That is, boosting TPR comes with a cost of slightly increasing FPR. When the FPR is higher than desirable, the algorithm stops.

5.3 Scalability. We ran the Polonium algorithm on the complete bipartite graph with 37 billion edges. Each iteration took about 3 hours to complete on average (∼185min). The algorithm scales linearly with the number of edges in the graph ($O(|E|)$), thanks to its adaptation of the Belief Propagation algorithm. We empirically evaluated this by running the algorithm on the full graph of over 37 billion edges, and on its smaller billion-edge subgraphs with around 20B, 11.5B, 4.4B and 0.7B edges. We plotted the per-iteration running times for these subgraphs in Figure 11, which shows that the running time empirically achieved linear scale-up.

5.4 Design and Optimizations. We implemented two optimizations that dramatically reduce both running time and storage requirement.

The first optimization eliminates the need to store the *edge file* in memory, which describes the graph structure, by externalizing it to disk. The edge file alone is over 200GB. We were able to do this only because the Polonium algorithm did not require random access to the edges and their associated messages; sequential access was sufficient. This same strategy may not apply readily to other algorithms.

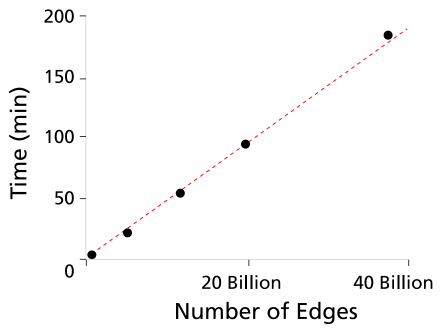


Figure 11: Scalability of Polonium. Running time per iteration is linear in the number of edges.

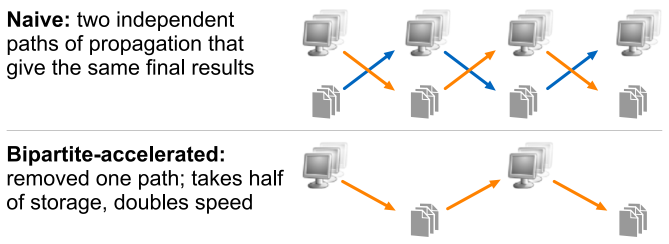

Naive: two independent paths of propagation that give the same final results
Bipartite-accelerated: removed one path; takes half of storage, doubles speed

Figure 12: Illustration of our optimization for the Polonium algorithm: since we have a bipartite graph (of *files* and *machines*), the naive version leads to two *independent* but *equivalent* paths of propagation of messages (orange, and blue arrows). Eliminating one path saves us half of the computation and storage for messages, with no loss of accuracy.

The second optimization exploits the fact that the graph is bipartite (of *machines* and *files*) to reduce both the storage and computation for messages by half [29]. We briefly explains this optimization here. Let $B_M[i,j](t)$ be the matrix of beliefs (for machine i and state j), at time t, and similarly $B_F[i,j](t)$ for the matrix of beliefs for the files. Because the graph is bipartite, we have

$$B_M[i,j](t) = B_F[i',j'](t-1) \tag{5.1}$$
$$B_F[i',j'](t) = B_M[i,j](t-1) \tag{5.2}$$

In short, the two equations are completely decoupled, as indicated by the orange and blue edges in Figure 12. Either stream of computations will arrive at the same results, so we can choose to use either one (say following the orange arrows), eventually saving half of the effort.

6 Significance and Impact.

In August 2010, the Polonium technology was deployed, joining Symantec's other malware detection technologies to protect computer users from malware. Polonium now serves 120 million people around the globe (at the end of September 2010). It has helped answer more than *one trillion* queries for file reputation.

Polonium's effectiveness **in the field** has been empirically measured by security experts at Symantec. They sampled live streams of files encountered by computer users, manually analyzed and labeled the files, then compared their expert verdicts with those given by Polonium. They concluded that Polonium significantly lifted the detection rate of a collection of existing proprietary methods by 10 *absolute* percentage points (while maintaining a false positive rate of 1%). This *in-the-field* evaluation is different from that performed over ground-truth data (described in Section 5), in that the files sampled (in the field) better exemplify the types of malware that computer users around the globe are currently exposed to.

Our work provided concrete evidence that Polonium works well in practice, and it has the following significance for the software security domain:

1. It radically transforms the important problem of malware detection, typically tackled with conventional signature-based methods, into a large-scale inference problem.

2. It exemplifies that graph mining and inference algorithms, such as our adaptation of Belief Propagation, can effectively unearth malware.

3. It demonstrates that our method's detection effectiveness can be carried over from large-scale "lab study" to real tests "in the wild".

7 Discussion.

Handling the Influx of Data. The amount of raw data that Polonium works with has almost doubled over the course of about 8 months, now exceeding *110 terabytes*. Fortunately, Polonium's time- and space-complexity both scale linearly in the number of edges. However, we may be able to further reduce these requirements by applying existing research. Gonzalez et. al [30] have developed a parallelized version of Belief Propagation that runs on a multi-core, shared-memory framework, which unfortunately precludes us from readily applying it on our problem, as our current graph does not fit in memory.

Another possibility is to concurrently run multiple instances of our algorithm, one on each component of our graph. To test this method, we implemented a single-machine version of the *connected component* algorithm [31] to find the components in our graph, whose distribution (*size* versus *count*) is shown in Figure 13; it follows the Power Law, echoing findings from previous research that studied million- and billion-node graphs [31, 32]. We see one giant component of almost 950 million nodes (highlighted in red), which accounts for 99.77% of the nodes in our graph. This means our prospective strategy of running the algorithm on separate components will only save us very little time, if any at all! It is, however, not too surprising that such a giant component exists, because most Windows computers uses similar subset of system files, and there are many popular applications that many of our users may use (e.g., web browsers). These *high-degree* files connect machines to form the dominant component.

Recent research in using multi-machine architectures (e.g., Apache Hadoop) as a scalable data mining and machine learning platform [31, 33] could be a viable solution to our rapidly increasing data size; the very recent work by Kang et. al [33] that introduced the Hadoop version of Belief Propagation is especially applicable.

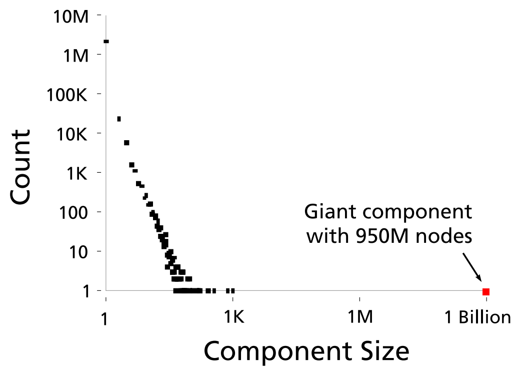


Figure 13: Component distribution of our file-machine bipartite graph, in log-log scale.

Perhaps, the simplest way to obtain the most substantial saving in computation time would be to simply run the algorithm for one iteration, as hinted by the diminishing return phenomenon observed in out multi-iteration results (in Section 5). This deliberate departure from running the algorithm until convergence inspires the optimization method that we discuss below.

Incremental Update of File & Machine Reputation. Ideally, Polonium will need to efficiently handle the arrival of new files and new machines, and it should be able to determine any file's reputation, whenever it is queried. The main idea is to approximate the file reputation, for fast query-time response, and replace the approximation with a more accurate value after a full run of the algorithm. Machine reputations can be updated in a similar fashion. The approximation depends on the maturity of a file. Here is one possibility:

Germinating. For a new file never seen before, or one that has only been reported by very few machines (e.g., fewer than 5), the Polonium algorithm would flag its reputation as "unknown" since there is too little information.

Maturing. As more machines report the file, Polonium starts to approximate the file's reputation through aggregating the reporting machines' reputations with one iteration of machine-to-file propagation; the approximation becomes increasingly accurate over time, and eventually stabilizes.

Ripening. When a file's reputation is close to stabilization, which can be determined statistically or heuristically, Polonium can "freeze" this reputation, and avoid recomputing it, even if new reports arrive. Future queries about that file will simply require looking up its reputation.

The NetProbe system [19], which uses Belief Propagation to spot fraudsters and accomplices on auction websites, used a similar method to perform incremental updates — the major difference is that we use a smaller induced subgraph consisting of a file and its direct neighbors (machines), instead of the 3-hop neighborhood used by NetProbe, which will include most of the nodes in our highly connected graph.

8 Future Work.

Using More Features. In this work, we only use a *subset* of all the data contributed by our users; similarly, the attributes mentioned in this paper are just a small subset of the vast number of machine- and file-based attributes that we have analyzed and leveraged to protect the users from security threats. By considering more attributes, we may obtain even better malware detection efficacy.

Weighing in File Prevalence and Correlation. All files are currently treated equally, no matter what their prevalence is. However, in reality, the cost of wrongly labeling a high-prevalence good file as bad has significantly higher cost than mislabeling a low-prevalence one. We may exploit the fact that some files (or applications) commonly exist together on a computer, to better estimate the reputations for these groups of files; alternative evaluation may then be performed at the *group level*, in addition to the current *file level*.

9 Conclusions.

In this paper, we motivated the need for alternative approaches to the classic problem of malware detection. We transformed it into a large-scale graph mining and inference problem, and we proposed the fast and scalable Polonium algorithm to solve it. Our goals were to infer the reputations of any files that computer users may encounter, and identify the ones with poor reputation (i.e., malware).

We performed a large-scale evaluation of our method over a real machine-file graph with one billion nodes and 37 billion edges constructed from the largest anonymized file submissions dataset ever published, spanning over *60 terabytes* of disk space. The results showed that Polonium attained a high true positive rate of 87.1% TPR, at 1% FPR. We also verified Polonium's effectiveness in the field; it has substantially lifted the detection rate of a collection of existing proprietary methods by 10 *absolute* percentage points.

We detailed important design and implementation features of our method, and we also discussed methods that could further speed up the algorithm and enable it to incrementally compute reputation for new files.

We believe our work is of considerable significance to the software security domain as it has demonstrated that the classic malware detection problem may be approached vastly differently, and could potentially be solved more effectively and efficiently; we offer Polonium as a promising solution. We also believe our work has brought great impact to computer users around the world, better protecting them from the harm of malware. Polonium is now serving 120 million people, at the time of writing. It has helped answer more than *one trillion* queries for file reputation.

10 Acknowledgements.

Duen Horng Chau was supported by the Symantec Research Labs Graduate Fellowship 2009–2010. We thank the many developers and engineers at Symantec who implemented and tested Polonium for production use. In particular, we thank Zulfikar Ramzan, Adam Bromwich, Vijay Seshadri and Daniel Asheghian for their helpful comments and suggestions. This material is based upon work supported by the National Science Foundation under Grants No. CNS-0721736 This work is also partially supported by an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

References

[1] Symantec. (2008, April) Symantec internet security threat report. [Online]. Available: http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf

[2] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proceedings of the 2003 ACM workshop on Rapid Malcode*. ACM New York, NY, USA, 2003, pp. 11–18.

[3] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, "Semantics-Aware Malware Detection," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2005, p. 46.

[4] Symantec. Malware definition. [Online]. Available: www.symantec.com/norton/security_response/malware.jsp

[5] N. Idika and A. P. Mathur, “A Survey of Malware Detection Techniques,” Department of Computer Science, Purdue University, Tech. Rep., 2007.

[6] M. Siddiqui, M. C. Wang, and J. Lee, “A survey of data mining techniques for malware detection using file features,” in *ACMSE ‘08*. New York, NY, USA: ACM, 2008, pp. 509–510.

[7] J. Kephart and W. Arnold, “Automatic extraction of computer virus signatures,” in *4th Virus Bulletin International Conference*, 1994, pp. 178–184.

[8] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, “Data mining methods for detection of new malicious executables,” in *IEEE Symposium on Security and Privacy*. IEEE COMPUTER SOCIETY, 2001, pp. 38–49.

[9] G. Tesauro, J. Kephart, and G. Sorkin, “Neural networks for computer virus recognition,” *IEEE expert*, vol. 11, no. 4, pp. 5–6, 1996.

[10] J. Kolter and M. Maloof, “Learning to detect and classify malicious executables in the wild,” *The Journal of Machine Learning Research*, vol. 7, p. 2744, 2006.

[11] S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine,” *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[12] J. Kleinberg, “Authoritative sources in a hyperlinked environment,” *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[13] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, “Combating web spam with trustrank,” in *VLDB ‘04*. VLDB Endowment, 2004, p. 587.

[14] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, “Fast best-effort pattern matching in large attributed graphs,” in *SIGKDD ‘07*. ACM, 2007, p. 746.

[15] X. Zhu, “Semi-supervised learning with graphs,” 2005.

[16] J. Neville and D. Jensen, “Collective Classification with Relational Dependency Networks,” in *Workshop on Multi-Relational Data Mining (MRDM-2003)*, p. 77.

[17] O. Neville, J. and şimşek, D. Jensen, J. Komoroske, K. Palmer, and H. Goldberg, “Using relational knowledge discovery to prevent securities fraud,” in *SIGKDD ‘05*. ACM, 2005, p. 458.

[18] M. Christodorescu, S. Jha, and C. Kruegel, “Mining specifications of malicious behavior,” in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007, pp. 5–14.

[19] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos, “Netprobe: a fast and scalable system for fraud detection in online auction networks,” in *WWW ‘07*. New York, NY, USA: ACM, 2007, pp. 201–210.

[20] M. McGlohon, S. Bay, M. Anderle, D. Steier, and C. Faloutsos, “SNARE: a link analytic system for graph labeling and risk detection,” in *SIGKDD ‘09*. ACM New York, NY, USA, 2009, pp. 1265–1274.

[21] J. Yedidia, W. Freeman, and Y. Weiss, “Understanding belief propagation and its generalizations,” *Exploring artificial intelligence in the new millennium*, vol. 8, pp. 236–239, 2003.

[22] R. Behrman and K. Carley, “Modeling the structure and effectiveness of intelligence organizations: Dynamic information flow simulation.” in *Proceedings of the 8th International Command and Control Research and Technology Symposium.*, 2003. [Online]. Available: http://www.casos.cs.cmu.edu/publications/papers/behrman_2003_modelingstructure.pdf

[23] S. A. Macskassy and F. Provost, “Suspicion scoring based on guilt-by-association, collective inference, and focused data access.” in *Proceedings of the NAACSOS Conference*, June 2005.

[24] X. Yan and J. Han, “gspan: Graph-based substructure pattern mining,” in *ICDM ‘02*. Washington, DC, USA: IEEE Computer Society, 2002, p. 721.

[25] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han, “Graphminer: a structural pattern-mining system for large disk-based graph databases and its applications,” in *SIGMOD ‘05*. ACM, 2005, p. 881.

[26] J. Pei, D. Jiang, and A. Zhang, “On mining cross-graph quasi-cliques,” in *SIGKDD ‘05*. ACM, 2005, p. 238.

[27] X. Yan, X. Zhou, and J. Han, “Mining closed relational graphs with connectivity constraints,” in *SIGKDD ‘05*. ACM, 2005, p. 333.

[28] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, “Coherent closed quasi-clique discovery from large dense graph databases,” in *SIGKDD ‘06*. ACM, 2006, p. 802.

[29] P. Felzenszwalb and D. Huttenlocher, “Efficient belief propagation for early vision,” *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.

[30] J. Gonzalez, Y. Low, and C. Guestrin, “Residual splash for optimally parallelizing belief propagation.” AISTATS, 2009.

[31] U. Kang, C. Tsourakakis, and C. Faloutsos, “PEGASUS: A Peta-Scale Graph Mining System,” in *ICDM ‘09*. IEEE, 2009, pp. 229–238.

[32] M. Mcglohon, L. Akoglu, and C. Faloutsos, “Weighted graphs and disconnected components: Patterns and a generator,” in *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIG-KDD)*, August 2008.

[33] U. Kang, D. Chau, and C. Faloutsos, “Inference of beliefs on billion-scale graphs,” *The 2nd Workshop on Large-scale Data Mining: Theory and Applications*, 2010.