Transductive Malware Label Propagation: Find Your Lineage From Your Neighbors

Deguang Kong
Dept of Comp. Sci. & Eng.
University of Texas at Arlington
Arlington, TX 76019
doogkong@gmail.com

Guanhua Yan
Information Sciences Group
Los Alamos National Laboratory
Los Alamos, NM, 87544
ghyan@lanl.gov

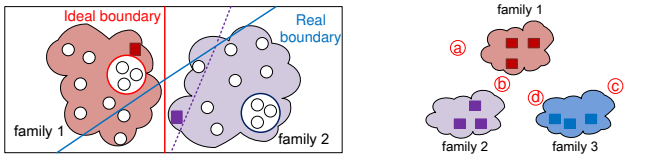
***Abstract*—The numerous malware variants existing in the cyberspace have posed severe threats to its security. Supervised learning techniques have been applied to automate the process of classifying malware variants. Supervised learning, however, suffers in situations where we have only scarce labeled malware samples. In this work, we propose a transductive malware classification framework, which propagates label information from labeled instances to unlabeled ones. We improve the existing Harmonic function approach based on the maximum confidence principle. We apply this framework on the structural information collected from malware programs, and propose a PageRank-like algorithm to evaluate the distance between two malware programs. We evaluate the performance of our method against the standard Harmonic function method as well as two popular supervised learning techniques. Experimental results suggest that our method outperforms these existing approaches in classifying malware variants when only a small number of labeled samples are available.**

I. Introduction

The numerous malware variants existing in the cyberspace have posed severe threats to its security. The Internet Security Report by Symantec revealed that it blocked 5.5 billion malware in 2011, an 81% increase over 2010, and discovered 403 million new malware variants, a 41% increase over 2010 [1]. In contrast to the large number of malware variants, however, is that the majority of them originated from only a handful of malware lineages. It is found, for instance, that as few as 25 families accounted for more than 75% of malware variants, according to the 2006 Microsoft Security Intelligence report [2].

For purposes of studying common characteristics of malware variants in the same family, as well as predicting trends of their evolution, it is urgent to develop methods that can automatically classify a large number of malware variants into their corresponding families. Automated malware classification requires techniques that are more accurate than the error-prone signature-based approach, which is commonly deployed by existing AV (Anti-Virus) software, and more scalable than manual reverse engineering by malware experts. Naturally, *supervised learning* has been suggested for the task of automated malware classification [3], as in essence, supervised learning is using a set of labeled samples to train a model (or a classifier in parlance of machine learning), which is further relied on to predict the classes of new unobserved data automatically. Hence, supervised learning allows us to leverage malware experts' knowledge about existing malware instances and foretell which family a new malware variant belongs to, based on how similar it is to those labeled ones in a predefined feature space.

978-1-4799-3360-0/14/$31.00 ©2014 IEEE



(a) Transductive learning (b) Discriminative Selection

Fig. 1: (a) A motivating example that shows malware classification with limited training samples. Malware are from two families. Two labeled malware are marked as red and blue squares. The other unlabeled malware are shown as open circles. The *ideal boundary* (red line) indicates an ideal classification boundary between the two families, and the *real boundary* (blue line) is the one obtained from a supervised learning method. (b) Selection of discriminative data in balanced class expansion. Unlabeled malware: a, b, c, d.

Applying supervised learning for malware classification requires sufficient labeled malware samples to obtain an accurate estimate of the boundaries that divide different malware families. In practice, however, we are often faced with situations where labeled malware samples are scarce. This is because identification of malware lineages, a task often done by malware experts who reverse engineer malware executables or observe malware behaviors in a controlled environment, is time consuming and also demands advanced domain knowledge in malware analysis.

Against the backdrop that performance of supervised learning deteriorates when only scarce labeled malware samples are available, we introduce a new paradigm, *transductive malware classification*, to infer family information of malware variants under such circumstances. Transductive malware classification is built on the concept of transductive learning, originally introduced by Vapnik [4] to contrast with inductive learning which infers general rules (e.g., discriminative models) from observed training cases, which has close relations with manifold learning. Transductive malware classification is only aimed at propagating label information from observed labeled malware instances to observed unlabeled ones based on clustering structures formed by both labeled and unlabeled samples, instead of solving a more general problem (as done by supervised learning), which is to find a good model capable of predicting label information of unseen test instances. Transductive malware classification is particularly suitable for situations where we own only a small number of labeled malware samples but want to classify a large set of unlabeled instances already available. To our best knowledge, this is the *first* work that

applies transductive learning to classify malware variants into their corresponding families. Even so, we still manage to improve the performance of a standard transductive learning technique, the Harmonic Function's approach for the purpose of automated malware classification. The crux of this work is more about a new paradigm for malware classification, rather than a comparison of different transductive learning techniques.

We summarize the contributions of this work. **(1)** We present a general transductive malware classification framework, and propose an improved label propagation algorithm based on the maximum consistency principle to infer malware families of unlabeled malware instances from known samples. **(2)** To apply transductive malware classification on structural information inherent in malware programs, we compute pairwise malware distances using a PageRank-like algorithm [5] on function call graphs of malware programs. **(3)** We perform extensive experiments and demonstrate the performance of our proposed transductive malware classification framework against two widely used supervised learning methods as well as the original Harmonic function approach.

II. Problem statement and Motivation

Suppose we have $(n = n_\ell + n_u)$ malware instances in $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{n_\ell+n_u}$, where the first n_ℓ are labeled and the remaining ones are not. The labeled malware instance $\mathbf{x}_i$ belongs to family y_i where $1 \leq i \leq n_\ell$ and $y_i \in \{1, 2, ..., K\}$. In practice, we have $n_l \ll n_u$. Our goal is to learn the class labels of unlabeled malware instances in $\mathbf{X}$, i.e., $\{y_i\}_{i=n_\ell+1}^{n}$. Let $\mathbf{Y} \in \Re^{n\times K}$ be a class indicator matrix, $\mathbf{Y}_{ij} = 1$ if $\mathbf{x}_i$ is labeled as class $y_i = j$; and $\mathbf{Y}_{ij} = 0$ otherwise. [1]

The conventional wisdom is to apply *supervised learning* techniques to train a model based on training instances in $\{\mathbf{x}_i\}_{i=1}^{n_\ell}$ and their corresponding label information. This approach, however, suffers significantly when only a limited number of labeled malware samples are available. This is because without sufficient training data, the classifiers learned from them cannot predict the boundaries among instances in different malware families accurately. This is manifested in the example illustrated in Figure 1a.

Suppose there is only one labeled malware in each family. A supervised learning method can be easily misled to produce an incorrect decision boundary (blue line in Fig. 1a) far away from the ideal one (red line in Fig. 1a). The reason is that the limited labeled malware available can not capture the structure of all malware data. In contrast, using label propagation, we can spread the label information from the two labeled instances to those belonging to their respective clusters. In this way, although no model or classifier is explicitly trained from the labeled malware samples, we can still classify other unlabeled instances correctly.

In another case, although several (or more) labeled samples are known from each family, if these labeled malware are close in distance (e.g., those malware samples inside the red and blue circles shown in Fig. 1a), the decision boundary can be still misled (shown as the purple dashed line in Fig. 1a). This is because the labeled samples are densely concentrated in a small region and their distribution is thus far different from that of the entire malware dataset. For supervised learning to function correctly, we expect that the structure formed by those unlabeled test samples be similar to that by the labeled instances from which we train a discriminative model.

Note for the problem stated above, training a classifier able to classify unseen malware samples outside $\mathbf{X}$ is unnecessary. Instead, we can rely on the clustering structure (formed by both labeled and unlabeled data) to propagate label information from labeled instances to unlabeled ones. It is this intuition that drives us to develop a transductive malware classification framework.

III. Transductive Malware Classification

Our transductive malware classification framework works on any pairwise malware similarity matrix $\mathbf{W}$ in which each element measures the similarity between two malware instances in the data sample space, regardless of being labeled or not. The similarity of two malware programs can be evaluated on a number of factors, such as their PE header information, n-gram byte sequences, disassembly code, and dynamic execution traces. In the next section, we shall apply the transductive malware classification framework on the structural information of malware programs. Here, we introduce the rationale behind a generic framework that works on any pairwise malware similarity $\mathbf{W}$. Our transductive malware classification framework uses a label propagation algorithm that improves the existing Harmonic function approach [6].

A. Review on Harmonic function approach

The Harmonic function approach [6] views both labeled and unlabeled samples as vertices in a weighted graph, with edge weights ($\mathbf{W}$) encoding the similarities between different samples. Label information is propagated from labeled samples to unlabeled ones using the harmonic function. Let $\mathbf{W} = \begin{pmatrix} \mathbf{W}_{ll} & \mathbf{W}_{lu} \\ \mathbf{W}_{ul} & \mathbf{W}_{uu} \end{pmatrix}$, where $\mathbf{W}_{ll} \in \Re^{n_l\times n_l}$ represents the similarity among n_l labeled data, $\mathbf{W}_{lu} \in \Re^{n_l\times n_u}$ the similarity between n_l labeled data and n_u unlabeled data, $\mathbf{W}_{ul} \in \Re^{n_u\times n_l}$ the similarity between n_u unlabeled data with n_l labeled data, and $\mathbf{W}_{uu} \in \Re^{n_u\times n_u}$ the similarity among n_u unlabeled data. Let $\mathbf{D} = \mathrm{diag}(\mathbf{d}_i)$ be the node degree diagonal matrix with entries $\mathbf{d}_i = \sum_{j=1}^{n} \mathbf{W}_{ij}$, and $\mathbf{F} = [\mathbf{F}_l; \mathbf{F}_u] \in \Re^{n\times K}$ be the class indicator, where $\mathbf{F}_l \in \Re^{n_l\times K}$ is the class indicator for the first n_l labeled data and $\mathbf{F}_u \in \Re^{n_u\times K}$ is the predicted class indicator for those n_u unlabeled instances. Our goal is to predict the class indicator $\mathbf{F}_u$ for the n_u unlabeled instances, which is given by

$$\mathbf{F}_u = \mathbf{Q}_{ul}\mathbf{F}_l, \qquad \mathbf{Q}_{ul} = (\mathbf{D}_{uu} - \mathbf{W}_{uu})^{-1}\mathbf{W}_{ul}; \tag{1}$$

where $\mathbf{D}_{uu} \in \Re^{n_u\times n_u}$ is the node degree diagonal matrix for the n_u unlabeled instances in diagonal matrix $\mathbf{D}$, and $\mathbf{Q}$ is label propagation operator. Then for each unlabeled malware $\mathbf{x}_i$ is assigned a class label according to $k = \underset{1\leq j\leq c}{\arg\max} (\mathbf{F}_u)_{ij}$.

B. Problems of standard harmonic function algorithm

Note that in Eq. (1), given the similarity matrix for all n malware and label information of $\mathbf{F}_l$ for n_l labeled malware, we can label n_u unlabeled malware through $\mathbf{F}_u$ *in one shot*. However, some unlabeled malware may lie near labeled malware in the data manifold while others lie far away from the labeled data. Therefore, the confidence we have about the class labels obtained from the Harmonic function approach may vary significantly across different data points.

[1] In this paper, scalars are denoted by lower-case letters $(n, i, \cdots)$, vectors in bolded lower-case letters $(\mathbf{e}, \mathbf{u}, \cdots)$, and matrices in bolded upper-case letters $(\mathbf{X}, \mathbf{Y}, \mathbf{W}, \cdots)$.

Consider two unlabeled malware instances $\mathbf{x}_i$ and $\mathbf{x}_j$, whose score distributions are given by

$$[(\mathbf{F}_u)_{i1}, \cdots, (\mathbf{F}_u)_{ic}] = [0.1, 0.04, 0.8, 0.3],$$

$$[(\mathbf{F}_u)_{j1}, \cdots, (\mathbf{F}_u)_{jc}] = [0.2, 0.35, 0.48, 0.40]. \quad (2)$$

Although both malware $\mathbf{x}_i, \mathbf{x}_j$ are assigned to family 3 (because family 3 obtains the largest confidence value 0.8 or 0.48), the confidence levels of two label assignments are different: we have high confidence in assigning malware $\mathbf{x}_i$ to family 3 as $(\mathbf{F}_u)_{i3} = 0.8$ is much higher than those of the other families, but low confidence in assignment of $\mathbf{x}_j$ because $(\mathbf{F}_u)_{j3} = 0.48$ is only *marginally* higher than $(\mathbf{F}_u)_{j4} = 0.40$. In other words, for $\mathbf{x}_i$ the propagation score distribution has a sharp distribution while for $\mathbf{x}_j$ the propagation score distribution is relatively flat.

C. Maximum confidence harmonic function algorithm

Motivated by this observation, we break the actual label assignment process into a number of iterations. Under the aforementioned circumstance, we assign class label for $\mathbf{x}_i$ and move it to a pool of already-labeled malware, while deferring the decision for $\mathbf{x}_j$ in later rounds. As the pool of already-labeled data expands to the neighborhood of $\mathbf{x}_j$, the propagation score distribution for $\mathbf{x}_j$ is likely to become sharper and we thus have a higher confidence in assigning a label to $\mathbf{x}_j$.

Therefore, our algorithm, which is guided by the *maximum confidence principle*, consists of multiple label propagation steps, and at step t:

$$\mathbf{F}_u{}^t = \mathbf{Q}_{ul}^{t-1} \mathbf{F}_l{}^{t-1}, \quad (3)$$

where for $t = 1, 2, ...$, $\mathbf{F}_l{}^t \in \Re^{n_l \times K}$ is the label assignment matrix using the current available labeled data during propagation step t, and $\mathbf{Q}_{ul}^t$ is the label propagation computed according to Eq. (1) using the *current* labeled and unlabeled malware. In each label propagation step, we use the current labeled data matrix $\mathbf{F}_l{}^t$ to update the label prediction matrix $\mathbf{F}_u{}^t$. At the end of each propagation step, only those unlabeled data points whose class labels are confidently predicted are actually assigned class labels and moved into the pool of labeled data (`Lpool`). The rest of unlabeled data points remain in the pool of unlabeled data (`Upool`). The process repeats itself until all malware instances are assigned with class labels.

Due to class imbalance concern, the pool of labeled malware (`Lpool`) should have approximately the same number of new members for each malware family. Hence, our algorithm allows each malware family to add one new member after each propagation step. We call this *balanced class expansion* (BCE). One critical issue in a BCE step is how to select the malware with the *maximum confidence* for each class. To address this issue, consider an example illustrated in Fig. 1b. Ideally, we want to add malware a instead of malware b to family 1, because a is far away from both families 2 and 3, and although b is closer to family class 1 as compared to a, it is also close to family 2 and 3. Similarly, it is preferable to add malware c rather than d to family 3.

Motivated by the above observation, we introduce the *discriminative score computation* for choosing malware with maximum confidences. For each unlabeled malware $\mathbf{x}_i$, we compute the confidence scores according to Eq. (3) and then sort them in non-increasing order:$(\mathbf{F}_u)_{ik_1} \geq (\mathbf{F}_u)_{ik_2} \geq (\mathbf{F}_u)_{ik_3} \geq ....$

We consider the three classes with the highest scores, k_1, k_2, and k_3, and assign the malware to one of them. The discriminative score of class k_j (where $j = 1, 2, 3$) is

$$E(i, c_{k_j}) = (\mathbf{F}_u)_{ic_{k_j}}^2 \frac{\sum_{h=1}^{3} |(\mathbf{F}_u)_{ic_{k_j}} - (\mathbf{F}_u)_{ic_{k_h}}|}{\sqrt{(\mathbf{F}_u)_{ic_{k1}} + (\mathbf{F}_u)_{ic_{k2}} + (\mathbf{F}_u)_{ic_{k3}}}}. \quad (4)$$

The denominator provides a mild scale normalization. Without this term, the class with the largest $(\mathbf{F}_u)_{ik_j}$ score may dominate the score computation process. It is a heuristic approach, and from the experiments we observe that using the highest three classes tends to lead to good results. Note that these scores are computed once for each BCE step. For each unlabeled data point $\mathbf{x}_i$ in `Upool`, it is assigned to class k, which has the largest $(\mathbf{F}_u)_{ik}$ scores among all classes. For each family k, we select the malware $\mathbf{x}_i$, which has the largest discriminative score $E(\mathbf{x}_i, c_k)$ among all malware in `Upool` assigned to family k.

If the initial distribution is unbalanced, we try to balance the number of labelled malware samples in different families until no more samples can be added to a family. To take into consideration the prior distribution for different malware families, we multiply the score tuple of a malware family (i.e., $\mathbf{F}$ in Eq. (2)) with the reciprocal of the number of samples in this malware family so that the contribution from each family is approximately the same.

D. Detailed Algorithm

The details of the algorithm are presented in Algorithm 1. The performance of our algorithm depends on the number of iterations executed, which can be controlled by how many instances are added to each malware group in an iteration. The execution time of each iteration is dominated by the inversion of the similarity matrix. We admit that our algorithm takes more time than the original Harmonic function's approach, as in the latter, only one iteration and thus one matrix inversion are required. Our algorithm also takes more time than kNN, which uses only $O(n^2)$ time. For SVM, a significant amount of time is spent on training the classifier as it requires to solve a convex optimization problem; once the classifier is trained, the time spent on classifying test data depends on both the size of the support vector and the amount of unlabelled input data. The improved classification accuracy, however, offsets the extra execution time incurred to our algorithm as we shall see later.

IV. Transductive Classification on Structural Information of Malware

In this section, we discuss how to prepare the similarity matrix W for the transductive malware classification framework based on the structural information extracted from malware programs. We use an attributed function call graph (FCG) to represent the structural information contained in a malware program, as described next.

A. Construction of Attributed FCG

The FCG captures the calling relationship of an executable program, and in an FCG each vertex represents a local function. For each local function, we first translate it into an intermediate language and then extract six types of attributes from it. They include `opcode` (the frequency of appearances for each opcode), `API` (the number of times each library API function is called), `memory` (the number of memory reading and writing operations made in this function), `IO` (the number of I/O reading and writing operations), `Register` (the number of reading and writing operations on each register), and `Flag`

Algorithm 1 Maximum confidence label propagation

Input: labeled data $L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$, unlabeled data $U = \{\mathbf{x}_j\}_{j=\ell+1}^{\ell+u}$, T
Output: predicted class labels for unlabeled malware
Procedure:
1: $t = 0$
2: **while** $t < T$ & U is not empty **do**
3: Compute current label malware indicator $\mathbf{F}_l{}^t$ with labeled malware L
4: Compute label propagation operator $\mathbf{Q}_{ul}^t$ with Eq. (1) using current labeled malware L and unlabeled malware U
5: Compute the label indicator for all unlabeled malware $\mathbf{F}_u{}^{t+1}$ with Eq. (1)
6: **for** each unlabeled data **do**
7: Compute its corresponding discriminative score using Eq. (4)
8: **end for**
9: **for** $k = 1$ to K **do**
10: Search all unlabeled data whose 1st choice target class is k {Balanced class expansion}
11: **if** the above search result is not empty **then**
12: Pick the one with the largest discriminative score, add it to class k, remove it from U
13: **end if**
14: **end for**
15: Update labeled and unlabeled malware pools L and U. {new labeled data added to `Lpool`}
16: $t = t + 1$
17: **end while**

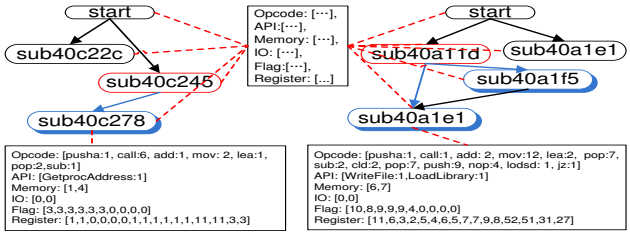


Fig. 2: Attributed FCGs of two malware samples. Each node represents a local function of disassembly code, where an edge represents the function call from one local function to another local function. Different attributes (such as API, register, memory information, etc) are collected over each function.

(the number of changes on each flag). For each attribute type, we represent it as a feature vector associated with the local function.

It is noted that a local function can further break down as a control flow graph, in which each vertex represents a basic block. For computational overhead concern, after extracting feature vectors for each basic block, we aggregate information of each attribute type by summing feature values over all basic blocks for each element in a feature vector. In this way, we do not need branch prediction, which is error-prone due to its heuristic nature.

B. Malware distance computation

Given the attributed FCGs of two malware programs, we now calculate their distance in terms of each attribute type as follows.

Step 1: Pairwise function distance computation First, we simply use the Euclidean distance to evaluate the distance between two nodes (i.e., local functions). Take Fig. 2 as an example. Let the `flag` attribute collected over the function node `sub40c245` in the first malware sample be $\mathbf{a} = [2, 1, 2, 2, 2, 2, 0, 0, 0, 0]$, and the `flag` attribute collected over the node `sub40a11d` in the second one be $\mathbf{b} = [1, 3, 1, 0, 1, 2, 0, 1, 1, 0]$. Then the distance between nodes `sub40c245` and `sub40a11d` w.r.t. `flag` attribute is $\sqrt{\sum_{i=1}^{10}(\mathbf{a}_i - \mathbf{b}_i)^2} = \sqrt{13}$.

Intuitively, the distance between two local functions is affected by not only how similar their feature vectors are but also whether the functions they call are also similar to each other. For example, in Fig. 2, the distance between `sub40c245` in $\mathcal{G}_1$ and `sub40a11d` in $\mathcal{G}_2$, is not only determined by their node distance, but also reflected by the distance between pairs (`sub40c278`, `sub40a1f5`) and (`sub40c278`, `sub40a1e1`), which are called by nodes `sub40c245` and `sub40a11d`, respectively.

Motivated by this observation, we update pairwise distances among local functions of the attributed FCGs of two malware programs using a PageRank-like algorithm. Let $A(x)$ denote the set of outgoing neighbors of node x. That is to say, for any local function x, $A(x)$ includes all the other functions called inside x. We also use vector $L^{in} \in \Re^n$ to include the in-degrees of all the nodes: if there are k edges coming into node x , then $L_x^{in} = k$.

Consider any attribute type. Let $D_0(\mathcal{G}_i, \mathcal{G}_j) \in \Re^{|\mathcal{V}_i| \times |\mathcal{V}_j|}$ denote the initial pairwise function distance matrix which includes only the Euclidean distances between the corresponding feature vectors of the function nodes in the attributed FCGs of malware i and j. For example, for Fig. 2, the initial distance matrix $D_0(\mathcal{G}_i, \mathcal{G}_j)$ is:

$$D_0 = \begin{pmatrix} start1 & 0.65 & 1 & 2.3 & 7 & 0.4 \\ 40c22c & 0.85 & 0.58 & 3.3 & 1.2 & 0.5 \\ 40c245 & 0.9 & 3.2 & 0.3 & 2.3 & 0.98 \\ 40c278 & 3.1 & 2.1 & 2.3 & 1.9 & 3.8 \\ & start2 & 40a11d & 40a1e1 & 40a1f5 & 40a1e1 \end{pmatrix},$$

where $D_0(start1, start2) = 0.65$ represents the initial function distance between function node `start_1` in $\mathcal{G}_i$ and function node `start_2` in $\mathcal{G}_j$. Let F_i^m be node m in the attributed FCG of malware i, and F_j^n be node n in the attributed FCG of malware j. Given all these notations, we update the pairwise function distance matrix: $D(F_i^m, F_j^n)$ as follows,

$$D(F_i^m, F_j^n) = (1-\alpha) D_0(F_i^m, F_j^m) + \alpha \sum_{\substack{n' \in W(n), \\ m' \in W(m)}} \frac{D(F_i^m, F_j^n)}{|L^{in}(m')||L^{in}(n')|}, \quad (5)$$

where α is a parameter that balances the effects of the Euclidean distances between feature vectors and the distances due to function calls (i.e., edges). Empirically, we set $\alpha = 0.10$. For the example shown in Fig. 2, let c= `sub40c245`, C = `sub40a11d`, d = `sub40c278`, D = `sub40a1f5`, E = `sub40a1e1`, then $D(c, C) = (1-\alpha)D_0(c, C) + \alpha[D(d, E) + D(e, E)]$, because $L^{in}(e) = L^{in}(d) = L^{in}(E) = 1$. The convergence of the above algorithm can be similarly proved as that of PageRank.

Step 2: Malware distance computation We first initialize the distance between $\mathcal{G}_i$ and $\mathcal{G}_j$, i.e., $d(\mathcal{G}_i, \mathcal{G}_j)$, to be 0. Given the pairwise function distance matrix $D(\mathcal{G}_i, \mathcal{G}_j)$, we identify the pair of function nodes with the smallest distance, i.e., find $(m', n') = \arg\min_{(m', n')} D(F_i^m, F_j^n)$. We then update malware distance $d(\mathcal{G}_i, \mathcal{G}_j) = d(\mathcal{G}_i, \mathcal{G}_j) + D(F_i^{m'}, F_j^{n'})$, and set the node distance $D(F_i^{m'}, F_j^{n'})$ to infinity in the pairwise function distance matrix. Hence, no node pairs which involve m' or n' will be considered again. Again, in the remaining pairwise node, we select the pair (m'', n'') $(m'' \neq m', n'' \neq n')$ with the smallest distance in the pairwise function distance matrix, $D(F_i, F_j)$, update malware distance $d(\mathcal{G}_i, \mathcal{G}_j) = d(\mathcal{G}_i, \mathcal{G}_j) + D(F_i^{m''}, F_j^{n''})$, and set $D(\mathcal{G}_i^{m''}, \mathcal{G}_j^{n''})$ to infinity. This process is repeated until no node pairs can be selected from $D(F_i, F_j)$. As the attributed FCGs $\mathcal{G}_i$ and $\mathcal{G}_j$ may have different numbers

of function nodes, there may be nodes that are left unmatched. Let V^i and V^j be the set of nodes in $\mathcal{G}_i$ and $\mathcal{G}_j$, respectively. Assuming that $|V^i| > |V^j|$, some nodes in $\mathcal{G}^i$ are not matched against any node in $\mathcal{G}^j$. In this case, we match these unmatched nodes against virtual nodes in $\mathcal{G}^j$, whose attributes are all set to $\mathbf{0}$, which means no functionality. Then the corresponding distance is added to $d(\mathcal{G}_i, \mathcal{G}_j)$. Finally, we return the value of $\frac{d(\mathcal{G}_i,\mathcal{G}_j)}{|V^i||V^j|}$ as the distance between malware $\mathcal{G}_i$ and $\mathcal{G}_j$. Note here the normalized term $|V^i||V^j|$ is used as the dominator to reduce the effect of the numbers of function nodes in the two attributed FCGs.

Step 3: Pairwise malware similarity computation Once we have computed the distances between all pairs of malware programs, we construct a pairwise malware similarity matrix $W \in \Re^{n\times n}$ (suppose we have n malware) using a standard Gaussian kernel. This is to say, the similarity between malware $\mathcal{G}_i$ and $\mathcal{G}_j$ is defined as

$$W_{ij} = \exp^{-\gamma \frac{D(\mathcal{G}_i,\mathcal{G}_j)^2}{t^2}}, \tag{6}$$

where γ is a parameter, and t is the average distance of the k-nearest neighbors for each malware, and $D(\mathcal{G}_i, \mathcal{G}_j)$ is the distance between malware $\mathcal{G}_i$ and $\mathcal{G}_j$. This pairwise malware similarity is computed for each attribute type.

V. Experimental Results

Malware dataset. In this work, we use a malware dataset from Offensive Computing [7], which contains 526,179 unique malware variants collected in the wild. We upload all our malware variants to the VirusTotal website [8], and get the classification results from 43 Anti-Virus software. As the malware dataset contains both packed and unpacked instances, in our evaluation, we only use unpacked malware, which we disassemble using IDA pro [9]. The reason we ignore packed malware instances is that the information we collect from them reflects only the functionality of their unpacking procedures rather than that of the malware programs themselves.

To validate the effectiveness of our approach, we need a malware dataset with known families. As reverse engineering each malware variant to figure out its lineage is a daunting task, we use the majority agreement results from five Anti-Virus Software (i.e., McAfee, NOD32, Kaspersky, Microsoft, and Symantec): if more than 3 of them classify a malware into the same family, we label this malware as a variant in this family. In this way, we identify instances from 12 families (`Bagle`, `Bifrose`, `Ldpinch`, `Swizzor`, `Zbot`, `Hupigon`, `Koobface`, `Lmir`, `Rbot`, `Rbot`, `Sdbot`, `Vundo`, and `Zlob`). These malware include worms, backdoor trojans, and also multi-component malware, which have diverse functionalities, such as stealing user data, connecting to remote IPs, establishing IRC communication, etc.

A. How biased is the dataset?

Before presenting the experimental results using our method, we first show the basic characteristics of those malware instances from the 12 families. Since we use majority agreement results from five Anti-Virus Software, a natural question is: *how biased is the dataset?* As sharply observed in [10], utilizing the concurrence of multiple anti-virus tools to produce ground-truth data may bias the dataset towards containing easy-to-cluster instances.

To answer this question, for each of the 12 malware families, we randomly select 100 instances, and we thus have 1200 malware instances. These 1200 malware are also used in the following classification experiments. Following the methodology introduced in Section IV, we compute the pariwise malware similarity using Eq. (6), and then feed the computed similarity to a state-of-the-art clustering algorithm, spectral clustering algorithm [11]. As in most of previous works [12], we use accuracy (ACC), normalized mutual information (NMI) and purity (PUR) to evaluate clustering results. The higher these values, the better the clustering results. Note it is actually *unsupervised learning*, and thus we use different measurements here from those when evaluating supervised or transductive learning methods. As shown in Fig. 3a, the maximum clustering accuracy is obtained by using `register` attribute. It is, however, no more than 62.35%, suggesting that the malware dataset is not clustered with high accuracy. One can get similar conclusions by analyzing the normalized mutual information or purity of clustering results. These statistics also indicate using the `register`, `opcode` and `API` attributes gives better clustering results than using the other attribute types.

TABLE I: Average F_1 measures using different methods over a range (i.e., $[5\%, 10\%, \cdots, 40\%]$) of labeled malware samples under **localized selection**. The similarity matrix in Eq. (6) is constructed when $k = 12, \gamma = 1.3$. The numbers are shown in percentages.

Classifier	Attribute-type					
	opcode	memory	register	io	flag	api
Har	51.24	50.19	59.07	46.85	40.53	54.56
kNN	44.04	40.09	47.99	37.46	31.02	39.12
SVM	45.36	41.67	45.30	36.62	30.76	44.26
M-Har	**55.26**	**52.82**	**67.27**	**48.52**	**43.79**	**55.25**

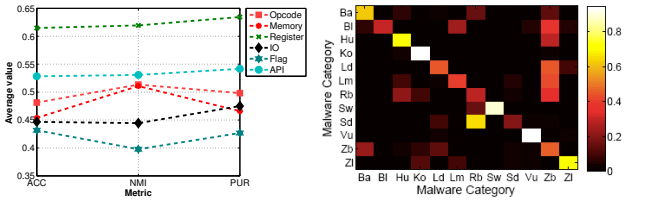
TABLE II: Average F_1 measures using different methods over a range (i.e., $[5\%, 10\%, \cdots, 40\%]$) of labeled malware samples under **random selection**. The similarity matrix in Eq. (6) is constructed when $k = 12, \gamma = 1.3$. The numbers are shown in percentages.

Classifier	Attribute-type					
	opcode	memory	register	io	flag	api
Har	52.75	51.94	59.52	47.75	41.72	56.83
kNN	47.54	48.95	57.05	41.87	38.57	46.51
SVM	46.04	45.61	50.93	43.69	41.78	44.23
M-Har	**57.09**	**54.94**	**66.90**	**49.80**	**44.80**	**58.60**

B. Maximum confidence malware classification

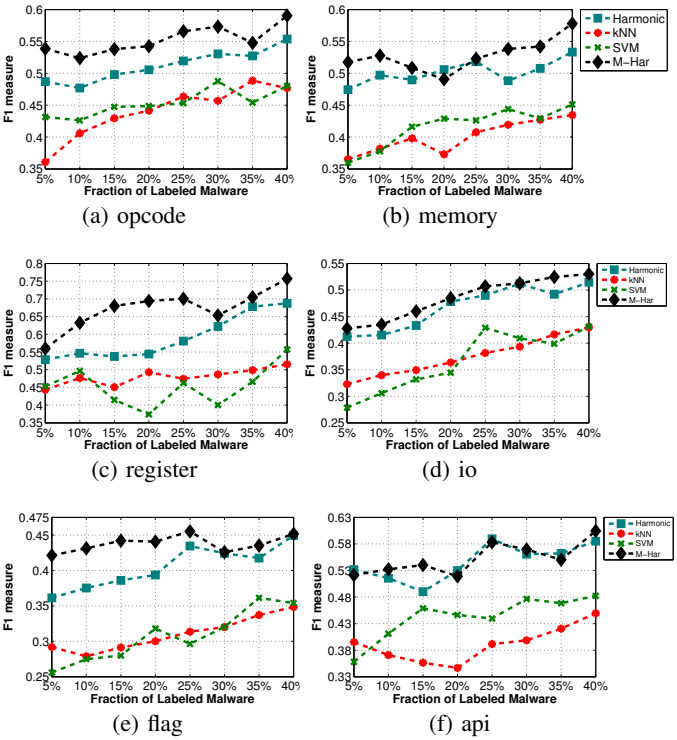
Following the methodology introduced before, we compute pairwise malware similarity using Eq. (6) for each type of attributes, and then feed them into four different transductive learning algorithms: proposed maximum confidence Harmonic function (`M-Har`), standard Harmonic function (`Har`) [6], and two standard supervised learning methods: k-nearest neighbor (`kNN`), and support vector machine (`SVM`). As KNN and SVM are two widely used supervised learning techniques, we use only them for comparison. A key challenge facing any supervised learning technique is that it requires a significant number of labelled samples to train a good classifier. This does not work well for malware classification, when only limited labelled samples are available in practice. When weighted KNN is used, the number of nearest neighbors k is consistently set to be 12. Actually, we observe that varying k within a reasonable range changes little the classification performance. For SVM, we use the same kernel as the one used for Har and M-Har Eq. (6). Given all these considerations, we ensure we are not comparing apples and oranges.

We vary the fraction of labeled malware among 5%, 10%,



(a) Clustering performance (b) Cross family analysis

Fig. 3: (a) Malware clustering results under different attribute types ($\gamma = 1.3$, $k = 12$). (b) Confusion matrix using the maximum confidence harmonic function approach, with 40% randomly labeled malware on the `register` attribute. Similarity matrix in Eq. (6) is constructed when $k = 12, \gamma = 1.3$.



(a) opcode (b) memory

(c) register (d) io

(e) flag (f) api

Fig. 4: Performance comparisons of classification methods with respect to each attribute type (opcode, memory, register, io, flag, api) when we vary the fraction of labeled data under the **localized selection** scheme. Compared methods: Original Harmonic function (Har), k nearest neighbor (kNN), support vector machine (SVM), and maximum confidence Harmonic function (M-Har). Similarity matrix in Eq. (6) is constructed when $k = 12, \gamma = 1.3$.

15%, 20%, $\cdots$, and 40% for each malware family. We also use cross validation to test the performances of different classifiers. For example, if we only have 5% of malware instances as labelled data, then we use the remaining 95% malware to test the model. This process is iterated for 20 times, and we report the averages of classification results. We consider two methods for selecting labeled malware samples: **(1) Random selection**: randomly select a certain percentage of labeled malware from each malware family. The selection process is iterated for a number of times according to the cross-validation rule. **(2) Localized selection**: Select a certain percentage of malware from each malware family, but they are always bounded in a small local region (like malware samples shown in red and blue circles in Fig. 1a). In our experiments, this is accomplished as follows: first we randomly select one malware sample from each family, and then select the one that is closest to any sample that has already been selected. This process is iterated until the desired percentage of labeled samples is achieved.

The performance of a classifier can be quantified with precision, recall, and F_1. Larger values of those metrics indicate better classification results.

C. Experiments with different percentages of labeled malware

Fig. 4 and Fig. 5 compare the average F_1 measures of different classification methods over the 12 malware families when we vary the fraction of labeled malware from 5% to 40% under the *localized selection* and the *random selection* schemes, respectively. For each classification method considered, the general trend is that its classification performance improves with the amount of labeled data available. This is plausible because for supervised learning more labeled data mean more data for training the classifiers to be built, and for transductive learning, with more labeled data, the label propagation process starts with more accurate label information. We summarize these results in Tables I and II, and observe the following:

(1) When a single attribute type is used, the maximum confidence harmonic function algorithm outperforms the original Harmonic function approach, and is significantly better than the two supervised learning methods. On average, the maximum confidence Harmonic function algorithm outperforms the standard Harmonic function approach, kNN, and SVM by 3.41%, 13.86%, and 13.15% under the localized selection scheme, respectively, and by 3.60%, 8.61%, and 9.97% under the random selection scheme, respectively. The sharp performance improvements of our method over the two standard supervised learning techniques confirm our intuition stated in Section II.

(2) The discriminative power of an attribute type is strongly correlated with its performance in clustering. For instance, we observe from both tables that under our proposed method (i.e., M-Har), using the `register` attribute type provides the best classification accuracy and using the `flag` attribute type leads to the worst classification performance. These are consistent with their relative performances for clustering as seen in Figure 3a. We are not sure which attribute would possess the most discriminative power, given a fixed parameter γ in different methods. There is no such dominant feature or attribute over all γ values we studied.

(3) Comparing the classification performances under the two different schemes of choosing initial labeled samples, we find that the performances of transductive learning methods do not change significantly. When we use the standard Harmonic function approach and our proposed method, the average F_1 measure improves under the random selection scheme by merely 1.35% and 1.54% over that under the localized selection scheme, respectively. For the two supervised learning methods, however, uniformly choosing initial labeled samples leads to better classification performance than the localized selection scheme. The average F_1 measure of kNN and SVM under the random scheme improves by 6.81% and 4.72%, respectively, over that under the localized selection scheme. These results confirm our intuition from Figure 1a. Using the localized selection scheme, the structure formed by selected labeled samples does not preserve the clustering structures formed by all malware samples, and it is thus difficult for a supervised learning technique to find the correct boundaries that separate different malware families. This contrasts with the random selection scheme, which keeps the clustering structure inherent in all malware samples among those selected labeled samples

(if there are a sufficient number of labeled samples chosen).

It is noted that when scarce labeled data are available, we do not expect that the classification performance is as good as that we can typically observe from evaluation of supervised learning techniques with a large amount of training data available. In five-fold cross validation, for instance, we use 80% of the data to train the classifier and only 20% of the data are used for testing. Consider a naive classification method, which randomly assigns a label from k classes to any malware instance. Suppose that there are an equal amount of n samples in each class. Both the expected recall and precision are $1/k$. Hence, the average F_1 measure is also $1/k$. In our case, $k = 12$ and the expected F_1 measure under this trivial classification method is only 0.083. This suggests that even when only scarce labeled malware samples are available, using the maximum confidence harmonic function algorithm leads to much more accurate classification than the trivial random classification method.

D. Cross family analysis

We are also interested in investigating which malware families are hard to classify, and whether there are some families similar to each other. Fig. 3b shows the confusion matrix obtained using the maximum confidence Harmonic function algorithm with 40% randomly labeled malware w.r.t. the `register` attribute. A confusion matrix is a tabular layout in which a column corresponds to a predicted class while a row a real class of instances. The (i, j) element of a confusion matrix shows the fraction of samples from class i are labeled as class j. From a confusion matrix, we learn how each malware family is correctly classified as itself from the diagonal elements of the matrix, as well as how a malware family is incorrectly classified to another family from an off-diagonal element.

From Figure 3b, we make the following observations. First, some malware families are easy to distinguish, such as `Swizzor`, `Vundo`, `Koobface`, etc. Second, some malware families tend to be recognized as other ones. For instance, malware instances in the `Sdbot` family are often mislabeled as from `Rbot`. This is because the development of `Rbot` was influenced by the source code of `Sdbot` published on the Internet [13]. These observations agree well with the results from our other studies on the same malware dataset.

E. Effects of parameter γ in Eq. (6)

The key parameters in Har and M-Har are the gamma parameter in pairwise similarity computation of Eq.(6). To understand the effects of parameter γ in Eq. (6) on the performances of different classification methods, we perform another set of experiments, in which we fix $k = 12$ to search for the nearest neighbors and choose γ among $[0.1, 0.4, 0.7, 1, 1.3, 1.6]$. Fig. 6 shows the average F_1 measures at different fractions of initial labeled data (in $[5\%, 10\%, \cdots, 40\%]$) against different γ values using the localized selection scheme. We observe that under different γ values, our proposed method not only consistently outperforms the original Harmonic function approach, but also leads to better performances than the two standard supervised learning techniques. These results further confirm our observations made in Section V-C. We can draw similar conclusions from the results under the random selection scheme. Due to space limit, we omit them here.

VI. Soft Transductive Malware Classification

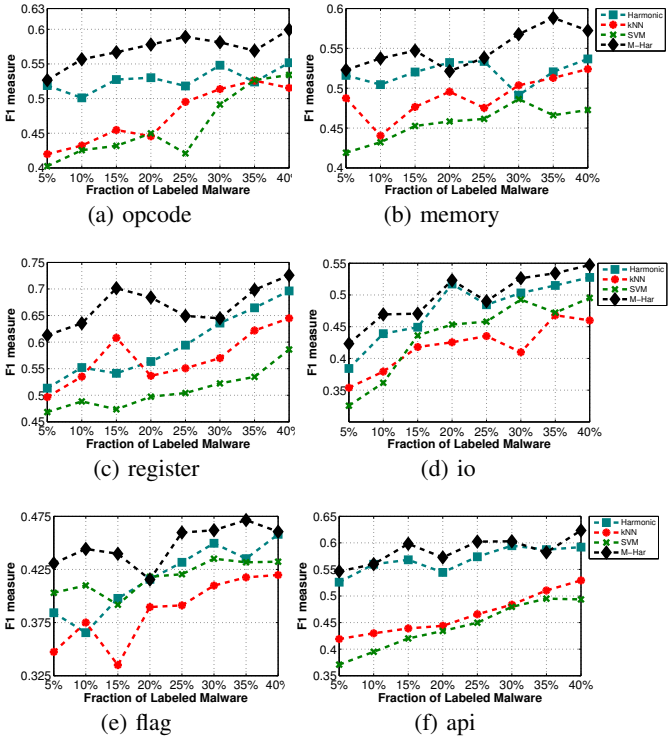
In previous sections, we have explored how to propagate label information from labeled samples to unlabeled ones, and this process terminates until *all* unlabeled samples are assigned a label. We term this *hard transductive malware classification*, as it requires all malware samples eventually to be labeled. In practice, however, hard transductive malware classification, as due to lack of labeled samples initially, some samples may be difficult to classify with only label information of existing labeled samples.

Motivated by this observation, we further adjust Algorithm 1 to provide a mechanism to balance the classification accuracy and the fraction of malware labeled eventually. Recall that in Algorithm 1, malware samples are moved from `Upool` to `Lpool` when they are assigned label information. In the soft version of transductive malware classification, we do so only if the samples can be labeled with a confidence level higher than a certain threshold. For malware i, we define the following: $D_i = \frac{|(\mathbf{F}_u)_{ik_1} - (\mathbf{F}_u)_{ik_2}|}{(\mathbf{F}_u)_{ik_1}}$. where we recall that $(\mathbf{F}_u)_{ik_1}$ and $(\mathbf{F}_u)_{ik_2}$ are the the highest two confidence scores of assigning unlabeled malware sample i to the existing malware families. Intuitively, D_i provides a metric to evaluate the sharpness for labeling malware i. The larger D_i is, the more confident we are in assigning the label of malware i. Hence, we use D_i to decide whether to label malware i: if $D_i < \theta$, where θ is a predefined threshold, we will not add malware i to the `Lpool`. The reason is the difference between assigning malware i to class k_1 and to class k_2 is not significant to make a confident decision.

Next we evaluate the effects of parameter θ on the accuracy of malware classification as well as the fraction of malware samples eventually labeled. In Fig. 7, with the 10% initially labeled malware, we show both the fractions of malware in the set of unlabeled 90% malware that are eventually labeled and the average F_1 measures when we vary θ among $[0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]$ under the *random selection* strategy. From the experimental results, we observe that as θ becomes larger, few malware samples are labeled eventually, but with higher accuracy on average. This agrees well with our intuition. Hence, if in practice a high classification accuracy is important, we can set θ to be a relatively high value. For instance, when the `register` attribute is considered, setting θ to be 0.6 enables us to label confidently 210 malware samples from a set of malware samples with only 10% of them originally labeled with the F_1 measure as high as 91.25%.

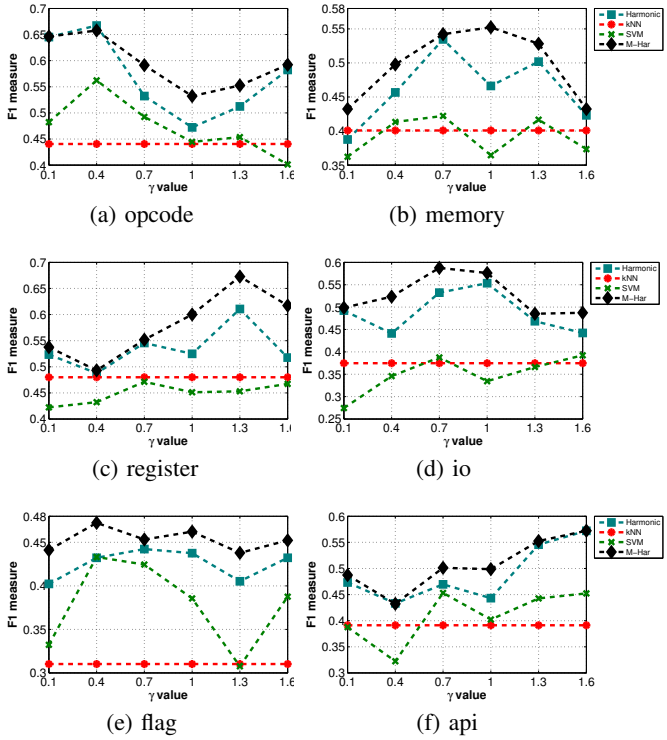
VII. Related work

Supervised malware classification. Supervised classification techniques have been widely used to classify an unknown file as a malware or a benign file (e.g., [14], [15], [16], [17], [18], [19], [20], [21], [22]), where a classifier is built using a large amount of labeled training samples with labels indicating whether they are malicious or not. In our work, instead of *only* making decisions on whether a malware is malicious or not, we aim at classifying malware into different families using structural information collected from malware samples with *limited* labeled malware. In contrast to these previous works that rely on supervised learning techniques to classify malware variants, our work presents a new malware classification paradigm that differs significantly from supervised malware classification. Features extracted from both static analysis and dynamic analysis are used for malware classification or malware detection, such as n-gram [23], function-call graph [24], [25], PE-header [26], and system-call [27], run time behaviors [28], [29], [30], etc. In our previous work, we compared the



(a) opcode (b) memory (c) register (d) io (e) flag (f) api

Fig. 5: Performance comparisons of classification methods with respect to each attribute type (opcode, memory, register, io, flag, api) when we vary the fraction of labeled data under the **random selection** scheme. Compared methods: Original Harmonic function (Har), k nearest neighbor (kNN), support vector machine (SVM), and maximum confidence Harmonic function (M-Har). The similarity matrix in Eq. (6) is constructed when $k = 12, \gamma = 1.3$.



(a) opcode (b) memory (c) register (d) io (e) flag (f) api

Fig. 6: Performance comparisons of classification methods with respect to each attribute type (opcode, memory, register, io, flag, api, and mixture) when we vary the fraction of initial labeled samples among $[5\%, 10\%, \cdots, 40\%]$ under the localized selection scheme. Compared methods: Original Harmonic function (Har), k nearest neighbor (kNN), support vector machine (SVM), maximum confidence Harmonic function (M-Har). The similarity matrix in Eq. (6) is constructed under different γ values.

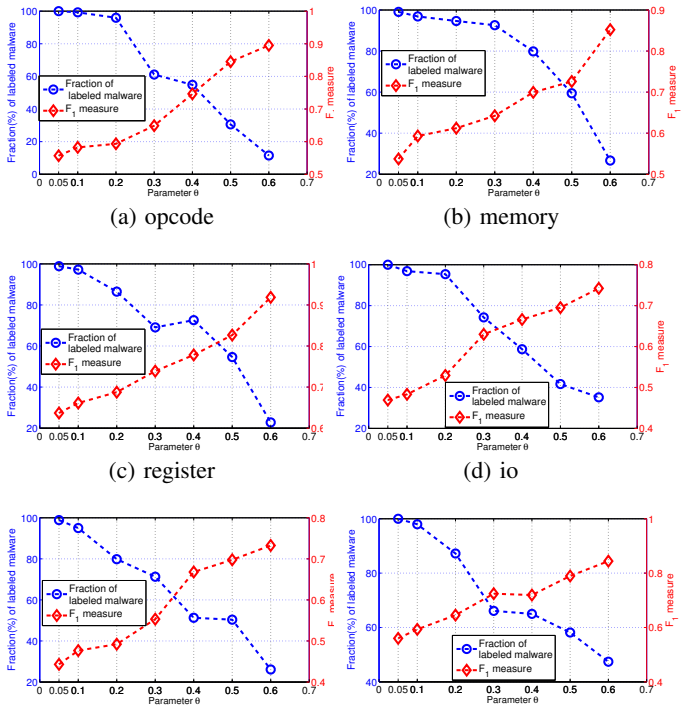
discriminative power of different types of malware features that can be represented as vectors of numerical values [31]. In this work, we extract malware features based on the FCGs obtained through static analysis. However, our proposed transductive classification can be used as a general framework for malware label propagation, and the other types of features (e.g., system call sequence) that have been considered in these previous works can be easily incorporated into our transductive malware classification framework.

Malware detection based on transductive learning. Little attention has been paid to malware classification using semi-supervised learning techniques except [32]. In [32], Santos et al. proposed to use the standard LGC (Local and Global Consistency) method [33] to detect unknown malware. The key differences between our work and theirs are as follows. First, our focus is not malware detection, which aims to distinguish malware from benign programs. Rather, our goal is to classify malware variants into their corresponding families. Second, their work relies on the byte n-gram representation of an executable program, which is amenable to existing classification methods as features are represented as vectors of numerical values. Structural information of an executable program, however, presents a new challenge in evaluating similarity among malware variants. To overcome this challenge, we have proposed a PageRank-like algorithm to estimate malware distances. For the purpose of transductive malware classification, we improve an existing transductive learning technique, i.e., the Harmonic function approach. Although comparison of its performance against other existing transductive learning techniques such as LGC [33] and Green's function [34] is beyond the scope of this work, our experimental results show that our proposed method stands out as a viable approach to transductive malware classification when only a small number of labeled samples are available.

Malware clustering. In contrast to malware classification, malware clustering applies to only those situations when no label information is available. The goal of malware clustering is to cluster different malware samples into different groups according to the inherent similarities/distances among malware instances. The malware clustering results can be further used for malware signature generation and other malware analysis. A large number of efforts have been dedicated to this direction, such as [35], [21], [36], [37], [10], [38], etc. Although our work has a different goal, which is to classify malware variants into their corresponding families when a limited number of labeled instances are available, the method we have proposed to measure the distances among malware programs can also be used for malware clustering.

VIII. Conclusions

Given the numerous malware variants on the Internet, it is urgent for us to develop effective, yet efficient, methods to classify them into their corresponding families. As supervised learning suffers when we do not have sufficient labeled samples to predict the class boundaries, we propose a new malware classification paradigm, transductive malware classification, which propagates label information from labeled instances to unlabeled ones. We apply this framework on the structural information collected from malware programs. Using extensive



(a) opcode (b) memory

(c) register (d) io

(e) flag (f) api

Fig. 7: Performance comparisons of classification methods with respect to each attribute type (opcode, memory, register, io, flag, api) using soft maximum harmonic function when we use **10%** labeled malware under the *random selection* scheme. The similarity matrix is constructed at $k = 12, \gamma = 0.3$.

experiments on a real-world malware dataset, we demonstrate that our proposed method outperforms existing methods in malware classification in situations where only a small number of labeled malware samples are available.

REFERENCES

[1] Symantec Incorporation, "Internet security threat report 2011 trends," vol. 17, 2012. [Online]. Available: http://www.symantec.com/threatreport

[2] Microsoft Incorporation, "Microsoft security intelligence report," http://orange.biolab.si/, January-June 2006.

[3] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *ACM AISec'11*.

[4] V. Vapnik, *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., 1982.

[5] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: bringing order to the web." 1999.

[6] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, 2003.

[7] Offensive Computing, http://www.offensivecomputing.net/.

[8] Virus Total, https://www.virustotal.com/.

[9] IDA Pro, http://www.hex-rays.com.

[10] P. Li, L. Liu, D. Gao, and M. K. Reiter, "On challenges in evaluating malware clustering," in *RAID*, 2010.

[11] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[12] D. Kong, C. H. Q. Ding, and H. Huang, "Robust nonnegative matrix factorization using l21-norm," in *CIKM*, 2011, pp. 673–682.

[13] Sdbot-rbot, http://www.honeynet.org/node/53.

[14] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *In Proceedings of the IEEE Symposium on Security and Privacy*, 2001, pp. 38–49.

[15] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.

[16] R. Perdisci, A. Lanzi, and W. Lee, "Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables," in *ACSAC*, 2008.

[17] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *SIGMETRICS*, 2013, pp. 347–348.

[18] Y. H. Park and D. S. Reeves, "Deriving common malware behavior through graph clustering," in *ASIACCS*, 2011, pp. 497–502.

[19] G. Jacob, R. Hund, C. Kruegel, and T. Holz, "Jackstraws: Picking command and control connections from bot traffic," in *USENIX Security Symposium*, 2011.

[20] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda, "Scalable, behavior-based malware clustering," in *NDSS*, 2009.

[21] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *RAID*, 2007, pp. 178–197.

[22] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *DIMVA*, 2008, pp. 108–125.

[23] Y. Jang, D. Brumley, and S. Venkatartaman, "Bitshed: feature hashing malware for scalable triage and semantic analysis," in *ACM Conference on Computer and Comunication Security*, 2011, pp. 309–320.

[24] X. Hu, T. cker Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs." in *CCS*, 2009.

[25] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Polymorphic worm detection using structural information of executables," in *RAID*. Springer-Verlag, 2005, pp. 207–226.

[26] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: Mining structural information to detect malicious executables in realtime," in *RAID*, 2009, pp. 121–141.

[27] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," in *ISSTA*, 2012, pp. 122–132.

[28] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *ESEC/SIGSOFT FSE*, 2007, pp. 5–14.

[29] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, "Synthesizing near-optimal malware specifications from suspicious behaviors," in *IEEE Symposium on Security and Privacy*, 2010, pp. 45–60.

[30] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. T. Giffin, and S. Jha, "Automatic generation of remediation procedures for malware infections," in *USENIX Security Symposium*, 2010.

[31] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'13)*, 2013.

[32] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in *DCAI*, 2011, pp. 415–422.

[33] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schlkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems 16*. MIT Press, 2004, pp. 321–328.

[34] C. H. Q. Ding, R. Jin, T. Li, and H. D. Simon, "A learning framework using green's function and kernel regularization with application to recommender system," in *KDD*, 2007, pp. 260–269.

[35] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda, "Scalable, behavior-based malware clustering," in *NDSS*, 2009.

[36] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *NSDI*, 2010, pp. 391–404.

[37] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[38] G. Yan, "Finding common ground among experts' opinions on data clustering: with applications in malware analysis," in *Proceedings of IEEE ICDE'14*, 2014.