Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

An Immune Local Concentration-Based Virus Detection Approach*

Wei WANG[1,2], Peng-tao ZHANG[1,2], Ying TAN‡[1,2], Xin-gui HE[1,2]

[1]*Key Laboratory of Machine Perception, Ministry of Eduction, Peking University, Beijing 100871, China*

[2]*Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University*

E-mail: ytan@pku.edu.cn

Received Dec. 26, 2010; Revision accepted Mar. 6, 2011; Crosschecked

Abstract: Along with the evolution of computer viruses, the number of file samples that need to be analyzed has constantly increased. An automatic and robust tool is needed to classify the file samples quickly and efficiently. Inspired by the human immune system, we developed a local concentration based virus detection method, which connects a certain number of two-element local concentration vectors as a feature vector. In contrast to the existing data mining technique, the new method does not remember exact file content for virus detection, but use a non-signature paradigm, such that it can detect some previously unknown viruses and overcome the techniques like obfuscation to bypass signatures. This model first extracts the viral tendency of each fragment and identifies a set of statical structural detectors, then uses an information-theoretic preprocess to remove redundancy in the detectors' set to generate 'self' and 'nonself' detector libraries. Finally, 'self' and 'nonself' local concentrations are constructed by using the libraries, to form a vector with an array of two elements of local concentrations for detecting viruses efficiently. Several standard data mining classifiers, including k-nearest neighbor (KNN), RBF neural networks, and support vector machine (SVM) are leveraged to classify the local concentration vector as the feature of a benign or malicious program and to verify the effectiveness and robustness of this approach. Experimental results demonstrate that the proposed approach not only has a much faster speed, but also gives around 98% of accuracy.

Key words: Local Concentration, Artificial Immune, Virus Detection

doi: **Document code:** A **CLC number:**

1 Introduction

Since the first malicious executable code appeared in 1981, computer viruses have been evolving with the rapid development of computer environments such as operating system, network, etc. There are three main virus detection methods: signature-based, malicious activity detection, and heuristic-based.

The natural immune system is a dynamic, adaptive, and distributed learning system. It protects organisms against antigen invasion by distinguishing foreign antigens (pathogens and tumor cells) from organismsą́ŕ own healthy cells and tissues and eliminating foreign antigens. Similarly, the functionality of computer security systems is to recognize and eliminate virus, so that the natural immune system has provided with an inspiration to develop such kind of an artificial immune based heuristic method for virus detection(Kephart, 1994).

In order to overcome the disadvantages, what of the widely used signature-based virus detection method, data mining and machine-learning approaches are also proposed for virus detection(Christodorescu *et al.*, 2007),(Kolter and Maloof, 2006). Many classification algorithms have been put into practice for solving virus problems combined with AIS including Naïve Bayes, Sup-

‡ Corresponding author

* This work was supported by the National Natural Science Foundation of China (Grant Nos. 60673020, 60875080), and partially supported by the National High-Tech Research & Development Program of China (Grant No. 2007AA01Z453).

port Vector Machine (SVM), Artificial Neural Network (ANN), and hybrid approaches(Christodorescu *et al.*, 2007),(Kolter and Maloof, 2006),(Schultz *et al.*, 2001), (Wang *et al.*, 2003).

In this paper, a novel AIS method is used to generate an array of two-element immune local concentration (LC) vectors as the feature vector for virus detection. 'Self' and 'nonself' detector libraries contain the bit strings are most representative of benign and virus programs, respectively. 'Self' and 'nonself' local concentrations are constructed by using 'self' and 'nonself' detector libraries to traverse the fixed length segment of a program. Then these two-element local concentrations of the program are connected to form a feature vector to identify a virus. The framework of the proposed technique is shown in Figure 1.

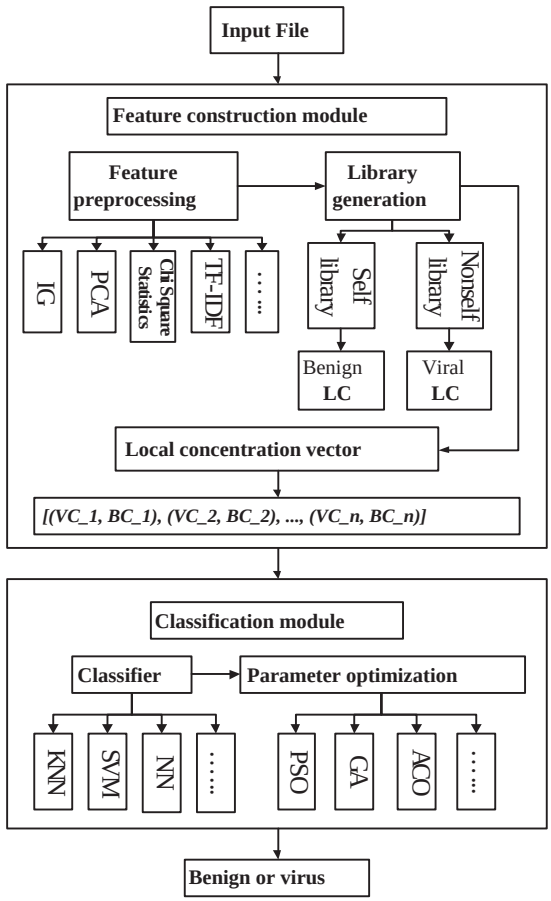

Fig. 1 Architecture of our proposed technique

Comprehensive experiments were conducted on a public virus data set in the previous works(Wang *et al.*, 2009),(Chao and Tan, 2009). Comparisons on performance were made against these works among different classifiers including k-nearest neighbor (KNN), RBF neural networks and support vector machine (SVM). Experimental results show that the proposed approach achieves more than 97% detection rate, and thus outperform the current approach. The runtime of training and detecting is relatively short. It takes 0.054 seconds on average to identify a given file.

It is well known that the signature-based virus detection method is incompetent to detect some new viruses. Furthermore, the number of malwares maintains an exponential growth, such that signature-based virus detection methods cannot keep pace with the security challenges, considering either increase of signatures' database or matching time of signatures. Differing from the existing data mining techniques including signature-based method and malicious activity detection, the proposed now approach neither memorizes specific byte-sequences appearing in the actual file content nor monitors suspicious program behavior. Our approach is non-signature based and therefore has the potential to detect previously unknown viruses. Moreover, in most of the current approaches, every selected detector is related to one feature dimension resulting in large dimensionality of feature. The proposed model reduces the feature dimensionality and extract position related information during the process of local concentration extraction. In this way, the model can somewhat resolve the two inherent shortcomings of non-signature based techniques: high false positive rate and large processing cost, resulting in low false positive rate and processing cost.

In this paper, Section II reviews the related work in detail. The generation of 'self' and 'nonself' detector libraries is described in section III followed by the construction of the feature vector shown in section IV. Experimental results and the conclusion are reported in section V.

2 Related work

As is mentioned above, there are three main feature construction approaches for virus detection.

The most common approach for virus detection is the signature-based method. It utilizes binary data mining to detect patterns in a large amount of data and use them to detect future instances in sim-

ilar data(Henchiri and Japkowicz, 2006). Because viruses can embed themselves in existing files, the entire file is searched not just as a whole, but also in pieces. This traditional method can detect a existing virus accurately, but it is somewhat limited by the fact that it can only identify a limited amount of generic or extremely broad signatures. It cannot detect some new emerging threats.

Malicious activity detection is another approach to identify viruses(Ilgun *et al.*, 1995),(Hofmeyr *et al.*, 1998). In this approach, the antivirus software monitors the system for suspicious program behaviors. If a suspicious behavior is detected, the suspect program may be further investigated, using signature-based detection or another method. This type of detection, by contrast, does not attempt to identify known viruses, but instead monitors the behavior of all programs. Unlike the signature-based approach, the suspicious behavior approach therefore provides protection against new viruses that do not yet exist in any virus dictionaries. Nevertheless, it is a dynamic form of monitoring, and the detection must rely on elements observable from an external agent. This method is often criticized because malicious actions are effectively executed(Kirda *et al.*, 2006) (Egele, 2008)(Jacob *et al.*, 2008).

Recently, heuristic methods that are more sophisticated, like malicious activity detection, are being actively investigated to identity unknown viruses. Since these methods operate at the byte-level file content, they do not require any a priori information about the viruses.

The most inspired heuristic virus detection method was proposed by Schultz *et al.*(Schultz *et al.*, 2001) and Kolter *et al.*(Kolter and Maloof, 2006).

The framework in(Schultz *et al.*, 2001) is composed of three learning algorithms: 1) An inductive rule-based learner that generates boolean rules based on feature attributes; 2) A probabilistic method generating the probability that an example was in a class given a set of features; 3) A multi-classifier system that combines the outputs from several classifiers to generate a prediction. These three independent techniques include system resource information, strings, and byte sequences extracted from the malicious executables in the data set as different types of features. The byte sequence technique, as used in our work, provides a relatively high detection accuracy. However, it requires large processing and memory which has been improved by Kolter *et al.* in(Kolter and Maloof, 2006).

Kolter *et al* used n-gram analysis and data mining to detect and classify malicious executables as they appear in the wild. The byte sequences were extracted from the executables, converted into n-grams, and the most relevant n-grams were treated as features. Their approach is evaluated for two aspects, including the classification between the benign and malicious executables and categorization of executables based on the function of their payload.

3 Generation of detector libraries

Algorithm 1 Algorithm for Virus Detection

Generate *'self'* and *'nonself' detector libraries* from training set
The sizes of the libraries are decided by parameter m which corresponds to proportional selection of the *potential detectors*

for each the sample in training set **do**
  Extract the *two-element local concentration vectors* of each segment in training sample through the two detector libraries
  Connect these local concentrations to an ordered feature vector as the input of a classifier
end for

Use these *feature vectors* to train a certain classifier

while Algorithm is running **do**
  if a program is detected **then**
    Characterize the sample by *local concentration vectors* through trained *'self'* and *'nonself' detector libraries*
    Use trained classifier to predict the *label* of the program
  end if
end while

The dimensionality N of the feature vector is decided by file truncated length

Our proposed approach is mainly divided into three parts: 1) Generation of 'self' and 'nonself' detector libraries from the randomly selected training

set; 2) Extraction of the two-element local concentration of each segment in a training sample and connect these local concentrations to construct a feature vector; 3) Three trained classifiers, including KNN, RBF neural networks and SVM, detecting the testing sample characterized using the ordered concentration vector. The overview of the proposed algorithm is outlined in Algorithm 1.

This approach computes a statistical and information-theoretic feature in a manner of local concentration on the byte-level file content. The generated feature vector of file segments is then given as an input to standard data mining classification algorithms that classify the file as virus or not.

The operating principle of generating 'self' detector library and 'nonself' detector library is shown in Figure 2. The concrete step is to divide all detectors into two sets by their tendency value and to calculate the detector importance, with important detectors retained.

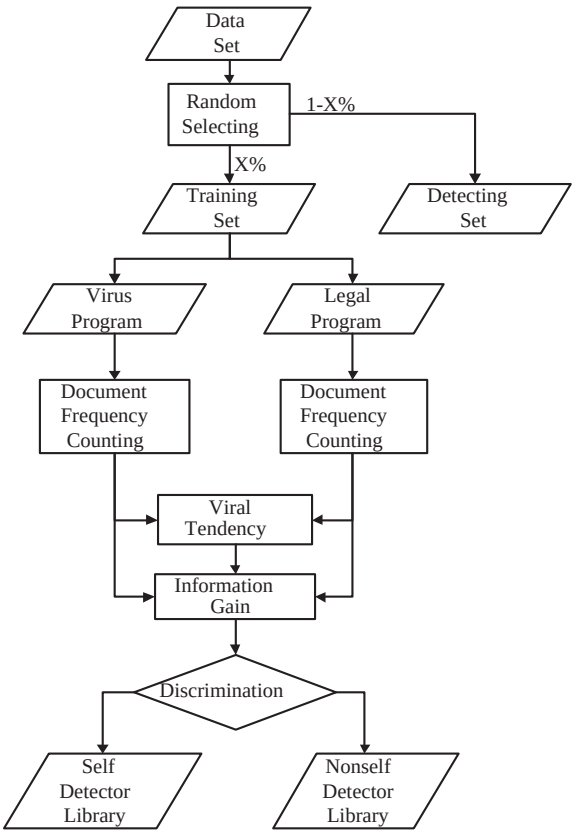

Fig. 2 Detector library generating process

'Self' detector library are composed of detectors most representative of benign files, and 'nonself' detector library are composed of those detectors most representative of viruses. Intuitively, the fragment that appears most frequently in virus programs, while rarely in benign programs, is a good representative of a virus.

The detectors in the library are a set of fixed-length fragments. Here a fixed length L-bit fragment of binary data, considered to contain appropriate information of functional behaviors, is taken as the detector to discriminate virus from benign program. The length L is set not too short to discriminate 'self' and 'nonself' and too long to make virus-specific data hidden in the binary data of files. Considering that one meaningful computer instruction is 8 or 16 bits normally, it is reasonable to set 'L' as 16, 32, or 64. A sliding window (shown in Figure 3, the overlap of sliding window is [L/2] bits) is used to count the document frequency of a detector in virus programs and benign programs. The difference of its document frequency in the virus programs and benign programs can reflect the tendency to be a virus or a benign file.

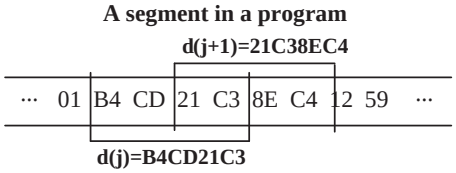

Fig. 3 Document frequency counting process, L=32bits

After counting the document frequency of each fragment, the tendency to be a virus $T(X)$ of fragment X is defined as in formula 1.

$$T(X) = P(X = 1|C_v) - P(X = 1|C_s) \qquad (1)$$

$P(X = 1|C_v)$ means document frequency of fragment X appearing in virus samples of training set; $P(X = 1|C_s)$ means document frequency of fragment X appearing in benign samples of training set. We defined the number of virus files as N_v, the number of benign files as N_s, the number of virus files which contain fragment X as n_v, and the number of benign files which contain fragment X as n_s, then:

$$P(X = 1|C_v) = \frac{n_v}{N_v} \qquad (2)$$

$$P(X = 1|C_s) = \frac{n_s}{N_s} \qquad (3)$$

If each fragment is extracted to form a dictionary, the size of this dictionary would become very large. The detectors appearing in most of files are not relevant to separate these files because all the classes have instances that contain these detectors. So with the number of detectors growing, the cost of computing would increase, but the effect may not be improved and made even worse. We reduce the number of fragments to generate 'self' and 'nonself' detector libraries according to different importance of each detectors. The importance of each detector is calculated based on Information Gain (IG). The detectors are sorted based on IG values in descending order, P% front of them are retained. Moreover, besides IG, other detector importance measures, such as document frequency, term-frequency variance, and χ^2 statistic, principal component analysis (PCA) can be applied to the model, endow it with promising development. The preprocess of statistical and information-theoretic feature generation is therefore achieved.

The generation of detector libraries is described in Algorithm 2, in which m is a parameter to be adjusted, and indicates proportional selection of all the fragments. The information gain is defined in formula 4.

$$IG(X,C) = \sum_{x\in\{0,1\},c\in\{C_v,C_s\}} P(X=x\wedge C=c) \cdot \log_2 \frac{P(X=x\wedge C=c)}{P(X=x)\cdot P(C=c)} \tag{4}$$

Where,
$P(X=1)$ means document frequency of fragment X which appears in the training set;
$P(X=0)$ means document frequency of fragment X which does not appear in the training set;
$P(C=C_v)$ means document frequency of virus files;
$P(C=C_s)$ means document frequency of benign files;
$P(X=0|C_v)$ means document frequency of fragment X which does not appear in virus samples of training set;
$P(X=0|C_s)$ means document frequency of fragment X which does not appear in benign samples of training set.

$$P(X=1) = \frac{n_v+n_c}{N_v+N_c} \tag{5}$$

$$P(X=0) = \frac{N_v+N_c-n_v-n_c}{N_v+N_c} \tag{6}$$

$$P(C=C_v) = \frac{N_v}{N_v+N_c} \tag{7}$$

$$P(C=C_s) = \frac{N_s}{N_v+N_c} \tag{8}$$

$$P(X=0|C_v) = \frac{N_v-n_v}{N_v} \tag{9}$$

$$P(X=0|C_s) = \frac{N_s-n_s}{N_s} \tag{10}$$

Algorithm 2 Algorithm for Generation of Detector Libraries

Initialize *'self'* and *'nonself' detector libraries* as $\varnothing$

while Algorithm is running **do**
  for each *fragment X* in the sample of training set **do**
    Calculate the *tendency* of *fragment X* by formula 1
    Calculate the *information gain* of *fragment X* by formula 2
  end for

  for each *fragment X* in the sample of training set **do**
    if $IG(X) > m$ **then**
      if $T(X) < 0$ **then**
        add *fragment X* into *'self' detector library*
      else
        add *fragment X* into *'nonself' detector library*
      end if
    end if
  end for
end while

Extract P% front of fragments to form *'self' detector library* and *'nonself' detector library*, P is decided by parameter m

4 Construction of feature vector

To construct a feature vector, a jumping window is moved to plot out several fixed length W-bit segments. Inside a fixed length W-bit segment in the program, a sliding window with [L/2] bits overlap is used to obtain the ąőselfąŕ local concentration and ąőnonselfąŕ local concentration (shown in Figure 4). In every window the local concentration of segment i is defined in formula 3 and 4.

$$VC_i = \frac{VN_i * L}{W} \tag{11}$$

$$BC_i = \frac{BN_i * L}{W} \tag{12}$$

Where VC_i and BC_i denotes the 'nonself' and 'self' local concentration, respectively; VN_i is the number of the detectors appearing in both detecting segment of the file and 'nonself' detector library; BN_i is the number of the detectors appearing in both detecting segment of the file and the 'self' detector library.

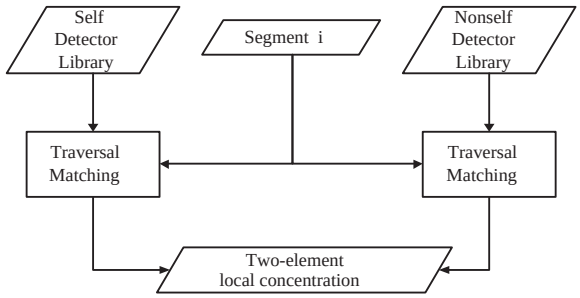

Fig. 4 Local concentration construction

After 'self' and 'nonself' local concentration are constructed in each window, these two-element local concentrations of the program are connected to form a feature vector $\langle(VC_1,BC_1),(VC_2,BC_2),\cdots,(VC_n,BC_n)\rangle$ (shown in Figure 5).

In order to serve these feature vectors as the input of successive classifiers for detecting, the dimensionality of the vector should be consistent. In this paper, truncated operation is applied and some rear dimensionality is discarded. We use $N * W$ bits information of each program, where N is the number of segments covered by the jumping window. The algorithm 3 is for feature construction.

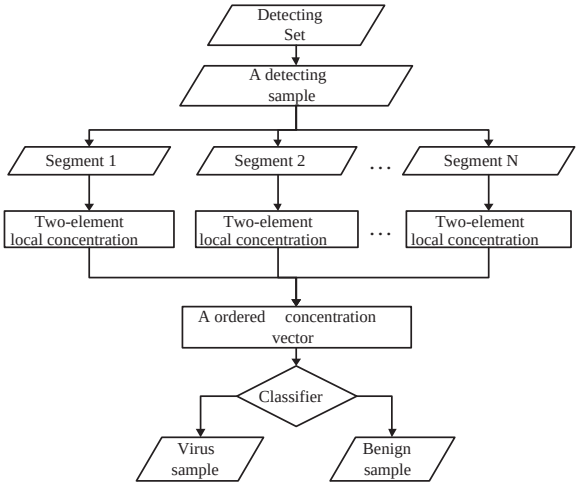

Fig. 5 Feature vector construction

Algorithm 3 Algorithm for Feature Construction

For a program to be detected, truncate front $N*W$ bits of the file and discard rear dimensionality of the file

for each segment inside *W-bit* jumping windows **do**
  Traverse the segment i using a *L-bit* sliding window with *[L/2] bits* overlap

  Initialize BN_i=0, VN_i=0
  for each different *L-bit fragment* in the segment i **do**
    if it appears in *'self' detector library* **then**
      BN_i++;
    else if it appears in *'nonself' detector library* **then**
      VN_i++;
    end if
  end for

  self-local concentration=BN_i*L/W
  nonself-local concentration=VN_i*L/W
end for

Connect these ordered two-element local concentrations to construct a feature vector

5 Data mining based classification

After characterizing the sample by a local concentration feature vector, one comes to the training phase with different classifiers. Here we introduce three simple but effective data mining methods used in this work.

5.1 KNN

KNN is a lazy learning method based on the closest training samples in the feature space. A sample is classified by a majority vote of its neighbors, with the sample being assigned to the class most frequent amongst its k nearest neighbors (k is a defined positive integer). During the classification phase, a detected feature vector is labeled by the class that is the most common among the k training samples nearest to that vector. Usually, Euclidean distance, Hamming distance and *etc.* can be used as the distance metric depended on different situation.

KNN is simple and effective. The classification with KNN is sensitive, however, to the data distribution. The disequilibrium distribution tends to influence the classification result: classes with more frequent samples tend to dominate the prediction of the new vector. Another problem is that during the training, all available data should be computed, which leads to considerable overhead when the training set is large.

In the KNN algorithm, the selection of k should be odd and small in order to avoid tie and misclassification. A good k can be optimized by some evolutionary algorithms.

5.2 SVM

A support vector machine is a supervised learning algorithm. The algorithm constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, maps the samples as points into a possibly high dimensional space and divides the samples into separate classes by a clear gap that is as wide as possible. An unlabeled sample is classified by the side of the separate hyperplane where the sample lies when it is mapped to the feature space.

SVM is a kind of generalized linear classifiers. In addtion, SVM can create non-linear classifiers with a non-linear kernel function instead of dot product by applying the kernel technique. This technique avoids the computational burden of explicitly representing the feature vectors.

In SVM training, cost parameter C and kernel parameters can influence the position of optimal hyperplane in the feature space and hence the performance of classification.

5.3 RBF Neural Network

Neural network is an adaptive system that its structure changes based on information that flows through the network during the learning phase, trying to simulate the functional aspects of biological neural networks.

Radial basis network is embedded in a neural network topology that uses radial basis function as activation function. Like the architecture of standard feedforward backpropagation network, it typically has three layers: an input layer, a hidden layer with a non-linear RBF activation function, and a linear output layer. In this paper, output layer is a sigmoid function of a linear combination of hidden layer values, representing a posterior probability, consisting of only one node– the label of the detected file.

Radial basis network works best when enough training vectors are available, and are more powerful in multidimensional space. It tends to have several times more neurons than a comparable feedforward network in the hidden layer and each neurons only respond to relatively small regions of the input space compared with standard neurons that output over a large region of the input space. The RBF network would not suffer from local minima as the error surface is quadratic. It is easy to find the minimum. Moreover, RBF network takes much less time than training a sigmoid/linear network(Chen *et al.*, 1991).

In the RBF network, the spread parameter controls the spread of the radial basis function. A larger spread leads to smoother radial basis function and more neurons responding to an input vector. A smaller spread leads to steeper radial basis function, so that the neuron with the weight vector closest to the input will have a much larger output than other neurons. The network tends to respond to the target vector associated with the nearest design input vector. It is necessary that the spread parameter to be large enough for the neurons to respond to overlapping regions of the input space, but not so large that all the neurons respond in essentially the same manner.

6 Experimental results and Conclusion

Experiments are conducted on a public virus data set in the pervious works(Wang *et al.*, 2009)(Chao and Tan, 2009). The "cilpku08" data set, is obtained from a famous virus Web site VX Heavens and from Computer-Forensic Experts of Anti-Virus Group in the Peking University, which can be get from the web site http://www.cil.pku.edu.cn/resources/. The folder includes 3547 malicious executables classified to 685 families based on their properties, comprising of six different types: virus, trojan, worm, backdoor, constructor, and miscellaneous. The most common file type for detection is virus, comprising more than 90% of all files; the remaining 10% of files are equally divided among the other five types. Our legal files are obtained from all folders of machines running the Windows 2000 and XP operating systems. This data set is divided into three subsets. The first data set contains 538 programs with the 'self' set of 284 legal files and the 'nonself' set of 254 virus files. The second data set contains 1815 programs with the 'self' set of 915 legal files and 900 virus files. The third data set consists of the second set and 2647 extra virus files, 4462 files in total. The training set is and much smaller than and covered by the detecting set, so that the expansibility and comprehensive ability can be tested.

The test platform for experiments is shown in Table 1.

6.1 Experiments for Different Window Size and Number of Windows

In this part, different window size W and number of windows N, correspond to dimension of the feature vector, are tested using three different classifiers, to find the parameters with the best performance. The tested W ranges from 100 to 500 with a step size 100 and N ranges 20 to 60 with a step size 10. The average results of ten experiments with different partitions of the second data set are used to measure the performance.

Figure 6,7,8 show that when W=400 and N=50 the results perform considerably stable and well on the data set. Thus these two parameters are fixed in the following of this paper.

6.2 Experiments for Different Proportional Selection of all the fragments

The size of the detector dictionary is decided by the proportional selection of the fragments. A suitable chosen proportional selection parameter m may reduce much the computing cost without losing its discriminatory power. The experiments for different m are also conducted on the second data set, m is chosen from 10% to 100% with a step size 10%. The results is shown in Figure 9.

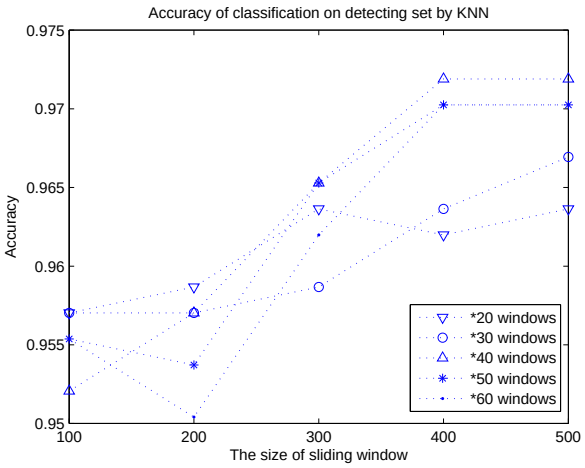

Fig. 6 Accuracy with different window size and number of windows on the second data set by KNN

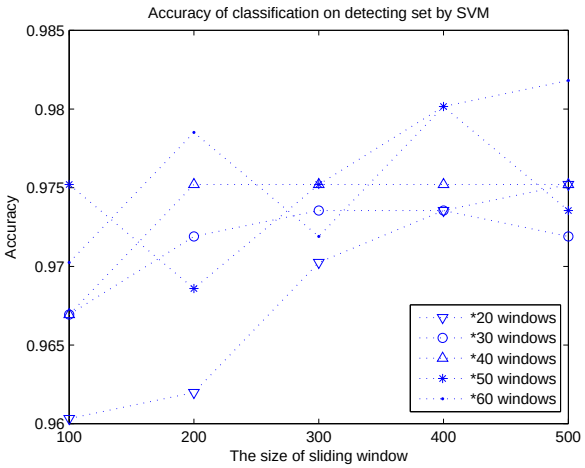

Fig. 7 Accuracy with different window size and number of windows on the second data set by SVM

When m=10%, the detecting rate has the best

Table 1 The test platform

Operating System	Windows XP
Computer Hardware	CPU: Pentium IV 1.5GHz RAM: 512M
Programming Language	C & Matlab language
Compiling Environment	Microsoft Visual C++ 6.0 & Matlab R2007a

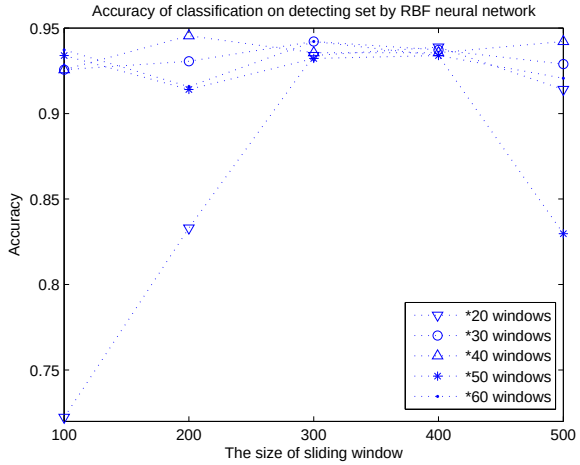

Fig. 8 Accuracy with different window size and number of windows on the second data set by RBF NN

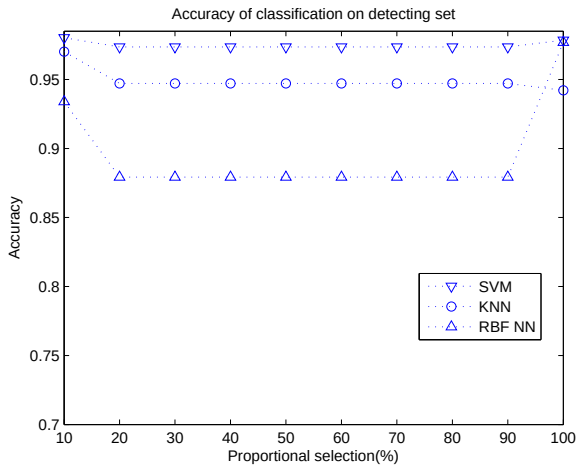

Fig. 9 Accuracy with different proportional selection of the fragments on the second data set

performance, where, the size of detector dictionary is smallest.

6.3 Length of the Detector

The length of the chosen detector L-bit is critical to discriminate viruses from benign programs. As the length of a meaningful program instruction is usually 16 bits, 32 bits, or 64 bits, L is not necessary bigger than 64 to contain at least one entire instruction. The length of the detector is taken as 32 or 64 in this paper in order to make it not too long to include some hidden viral information and not too short to obtain enough representative viral information. The overlap of sliding window is [L/2] bits. The accuracy rates using SVM classifier with different length of the detector are shown in table 2.

6.4 Contrast Experiments

In order to assess the performance and to show possible advantages of the proposed approach, nine contrast experiments against the method in (Wang *et al.*, 2009) were performed on these three practical data sets under windows operating system. The same partitions are made using the same data. Test 1, 2, and 3 were performed on the first data set with a partition ratio of 4:1,1:1,1:4 for the training set and the detecting set. Test 4, 5, and 6 were performed on it with a partition ratio of 2:1,1:1,1:2 for the training set and the detecting set. Test 7, 8, and 9 were performed on the second data set with a partition ratio of 2:1,1:1,1:2 for the training set and the detecting set.

As shown in Figure 10, and 11, our proposed method outclassed the hierarchical AIS method in all the tests, and achieved an accuracy rate of more than 97% on the detecting set. The proposed method did not appear to lose performance as the set size were growing. The runtime performance of our method also performed better than the hierarchical AIS method. In Figure 12, the training time of

Table 2 Average accuracy rate by SVM when L=64 or 32

Exp.	Accuracy rate with 64-bit Detector(%)						Accuracy rate with 32-bit Detector(%)					
	Training Set			Detecting Set			Training Set			Detecting Set		
	All	*Virus*	*Benign*	*All*	*Virus*	*Benign*	*All*	*Virus*	*Benign*	*All*	*Virus*	*Benign*
Test1	100.00	100.00	100.00	97.22	96.08	98.25	99.53	99.51	99.56	95.37	98.04	92.98
Test2	100.00	100.00	100.00	97.40	95.28	99.30	99.63	99.21	100.00	98.51	96.85	100.00
Test3	100.00	100.00	100.00	94.19	87.68	100.00	99.07	98.04	100.00	96.74	95.57	97.80
Test4	100.00	100.00	100.00	97.75	96.43	98.94	99.44	98.82	100.00	96.07	92.86	98.94
Test5	100.00	100.00	100.00	97.03	94.49	99.30	100.00	100.00	100.00	97.03	98.43	95.77
Test6	100.00	100.00	100.00	95.56	90.59	100.00	100.00	100.00	100.00	95.00	90.59	98.95
Test7	99.92	100.00	99.84	98.02	97.67	98.36	99.09	99.50	98.69	98.02	97.67	98.36
Test8	100.00	100.00	100.00	96.04	94.00	98.03	99.78	99.56	100.00	97.91	97.33	98.47
Test9	100.00	100.00	100.00	95.12	91.83	98.36	99.67	99.33	100.00	97.27	95.83	98.69

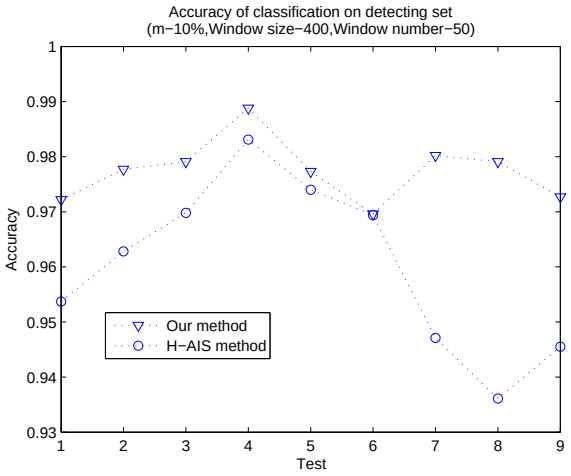

Fig. 10 Accuracy rate on the detecting sets of Contrast Experiments

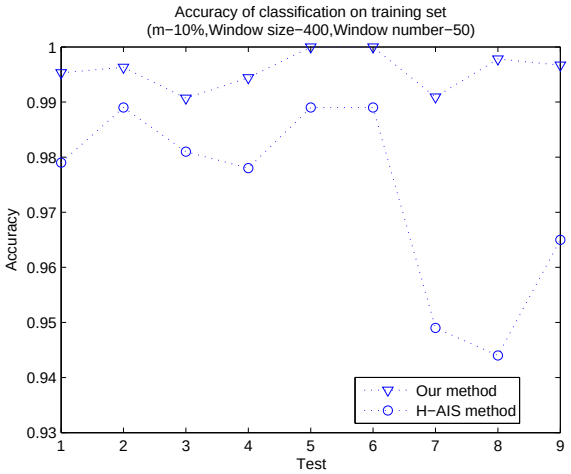

Fig. 11 Accuracy rate on the training sets of Contrast Experiments

new method varies linearly with the number of files, unlike H-AIS method whose training time grew exponentially with the file number. Furthermore, the runtime of new method (several minutes) and H-AIS method (several hours) are not of the same order of magnitude.

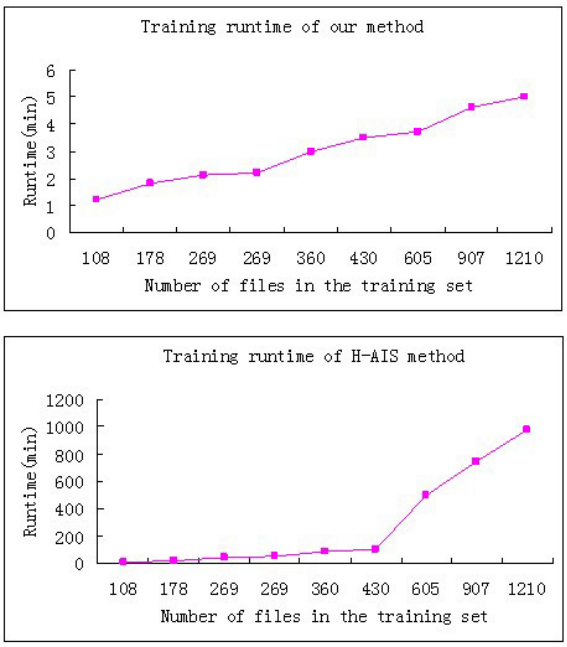

Fig. 12 Training runtime of two methods

Experiments on the third data set confirmed the model's expansibility, and the training set is much smaller than the detecting set. The results are shown in Figure 13.

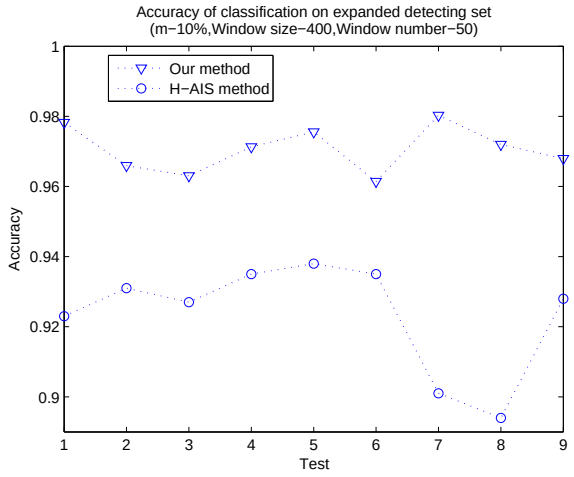

Fig. 13 Accuracy rate on the detecting sets of expanded Contrast Experiments by SVM

6.5 Parameters optimization

The feature vector constructed by an ordered two-element local concentration is the input of a classifier, and the binary value is the output. The generation of 'self' and 'nonself' detector libraries, the jumping window size W setting, the window number N setting, which in turn determine the feature vector, is here an optimization problem.

The vector that needs to optimize $P^* = \{m, W, N, P_1^*, P_2^*, \cdots, P_n^*\}$ is composed of the detector library determinant m, the jumping window size W, the window number N and the parameters $P_1^*, P_2^*, \cdots, P_m^*$ associated with a certain classifier.

When m is set to different values, P would take different values, and different detector libraries are obtained. An unique feature vector can be constructed when different W and N, for a file to be characterized, 'self' local concentrations that represent their similarity to benign program and 'nonself' local concentrations that represent their similarity to virus are different. $P_1^*, P_2^*, \cdots, P_m^*$ are classifier-related parameters that influence the performance of a certain classifier. Different classifiers hold different parameters and lead to varied performances. For examples, parameters associated with KNN include a number of nearest neighbors and the ways of distance measures. SVM-related parameters that determine the position of optimal hyperplane in feature space, include cost parameter C and kernel parameters.

The optimal vector is the one whose cost function associated with classification is minimum, namely the one which make the accuracy of classification maximum. The cost function $CF(P)$ can be defined as:

$$CF(P) = Err(P) \tag{13}$$

where $Err(P)$ is the classification error on the training set.

Input vector is composed of two parts: LC feature vector determinant m, W, N and $P_1^*, P_2^*, \cdots, P_m^*$ these classifier-related parameters. Output is to find a P^*, hence:

$$CF(P^*) = Err(P^*) = \min_{\{m,W,N,P_1^*,P_2^*,\cdots,P_m^*\}} Err(P) \tag{14}$$

Several robust optimization approaches can be employed to optimize the input vector, such as particle swarm optimization (PSO) and genetic algorithm (GA). Here we use a CPSO (as shown in Figure 14) to design the LC feature vector and the corresponding classifier. The detailed optimization process is referred to in (Tan and Xiao, 2007).

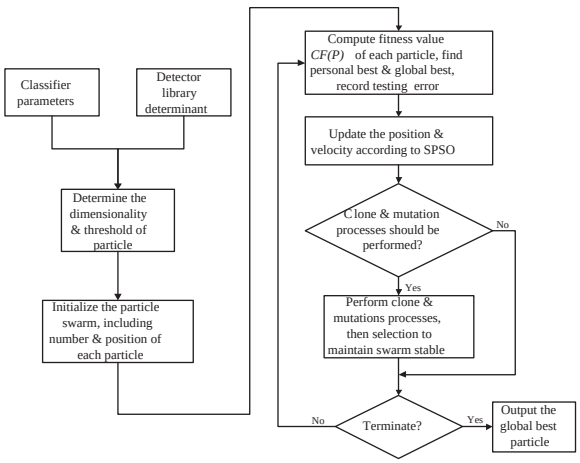

Fig. 14 CPSO-based classification process

The selection of the LC feature vector determinant m, W, N and the classifier-related parameters, $P_1^*, P_2^*, \cdots, P_m^*$, is a dynamic optimization process. Parameters associated with KNN include a number of nearest neighbors K and the ways of distance measures, K is optimized in the integer number interval [1, 20], and the ways of distance measures are chosen among *euclidean*, *cityblock*, *cosine*, *correlation*. For SVM, the cost parameter C is optimized in real number interval [1, 200]. For RBF neural network, the spread σ in real number interval [1, 5] is optimized. m is optimized in the integer number interval [5, 100], W, N range in [100, 600] and [10, 100],

Table 3 Average accuracy rates on the detecting set with empirical & optimized classification designs under optimum conditions by SVM

	Optimized designs		Empirical designs	
EXP.	*All*(%)	*FP*(%)	*All*(%)	*FP*(%)
Test1	97.83	4.37	97.83	4.37
Test2	98.51	0.00	96.59	2.95
Test3	96.74	2.20	96.30	3.61
Test4	97.75	1.06	97.13	3.17
Test5	97.56	3.93	97.56	3.93
Test6	96.15	3.17	96.15	3.17
Test7	98.03	1.64	98.03	1.64
Test8	97.91	1.53	97.20	1.09
Test9	97.27	1.31	96.80	1.20
Average	97.53	2.14	97.06	2.79

Table 4 Average accuracy rates on the detecting set with empirical & optimized classification designs under optimum conditions by KNN

	Optimized designs		Empirical designs	
EXP.	*All*(%)	*FP*(%)	*All*(%)	*FP*(%)
Test1	97.78	4.04	97.78	4.04
Test2	97.77	0.70	96.08	2.84
Test3	97.91	0.00	96.23	3.06
Test4	98.88	1.06	97.69	4.04
Test5	96.65	0.00	96.12	2.73
Test6	95.94	3.61	95.94	3.61
Test7	97.02	0.00	96.68	0.44
Test8	97.47	1.31	96.30	1.20
Test9	96.94	1.15	95.47	0.87
Average	97.37	1.32	96.48	2.54

respectively. The maximum number of generation is set to be 200 as the stop criterion, the number of particles in a swarm is 20.

The randomness of CPSO leads to slightly variation in the performance and obtained parameters. Therefore the results of nine independent classes of experiments on the expended third data set were used to evaluate tests, which is more reasonable. The average performance of empirical and optimized classification designs is reported in Table 3 & 4. FP means false positive rate, the rate of legal files mistakenly classified as malicious executables.

The results show that the optimized classification design resulted in a 1% increase in accuracy rate compared with the empirical classification design. The CPSO method has improved the accuracy and reduced the false positive rate. However, a trade-off decision has to be made between the better result and a much longer training time.

6.6 Conclusion

In this paper we have proposed an non-signature based approach that analyzes the byte-level file content. Instead of traditional binary data mining methods, our method first establishes a uniform framework for a general and systematic approach to feature construction. Second, it reduces the dimensionality resulting in faster training process. Also, the proposed feature extraction approach attains better or at least comparable results. The new method is easier without sacrificing performance, and provides implicit robustness against common obfuscation techniques.

References

Chao, Rui, Tan, Ying, 2009. a virus detection system based on artificial immune system. *Computational Intelligence and Security, International Conference on*, **1**:6-10. [doi:10.1109/CIS.2009.106]

Chen, S., Cowan, C.F.N., Grant, P.M., 1991. orthogonal least squares learning algorithm for radial basis function networks. *Neural Networks, IEEE Transactions on*, **2**(2):302 -309. [doi:10.1109/72.80341]

Christodorescu, Mihai, Jha, Somesh, Kruegel, Christopher, 2007. Mining specifications of malicious behavior. Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, New York, NY, USA, p.5–14, Available from http://doi.acm.org/10.1145/1287624.1287628. [doi:10.1145/1287624.1287628]

Deng, Peter Shaohua, Wang, Jau-Hwang, Shieh, Wen-Gong, Yen, Chin-Pin, Tung, Cheng-Tan, 2003. Intelligent automatic malicious code signatures extraction. Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on, p.600 - 603. [doi:10.1109/CCST.2003.1297626]

Egele, Manuel, 2008. Behavior-based spyware detection. VDM Verlag, Saarbrücken, Germany, Germany.

Henchiri, Olivier, Japkowicz, Nathalie, 2006. a feature selection and evaluation scheme for computer virus detection. *Data Mining, IEEE International Conference on*, **0**:891-895. [doi:10.1109/ICDM.2006.4]

Hofmeyr, Steven A., Forrest, Stephanie, Somayaji, Anil, 1998. intrusion detection using sequences of system calls. *Journal of Computer Security*, **6**:151–180.

Ilgun, K., Kemmerer, R.A., Porras, P.A., 1995. state transition analysis: a rule-based intrusion detection approach. *Software Engineering, IEEE Transactions on*, **21**(3):181 -199. [doi:10.1109/32.372146]

Jacob, GrÍęgoire, Debar, HervÍę, Filiol, Eric, 2008. behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, **4**:251-266 (10.1007/s11416-008-0086-0). Available from 10.1007/s11416-008-0086-0

Kephart, Jeffrey O., 1994. A biologically inspired immune system for computers. Proceedings of the Fourth International Workshop on Synthesis and Simulatoin of Living Systems, p.130–139, Available from http://jmvidal.cse.sc.edu/library/kephart94a.pdf

Kirda, Engin, Kruegel, Christopher, Banks, Greg, Vigna, Giovanni, Kemmerer, Richard A., 2006. Behavior-based spyware detection. In Usenix Security Symposium.

Kolter, J. Zico, Maloof, Marcus A., 2006. learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.*, **7**:2721–2744.

Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y., 2008. Unknown malcode detection via text categorization and the imbalance problem. Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on, p.156 -161. [doi:10.1109/ISI.2008.4565046]

Preda, Mila Dalla, Christodorescu, Mihai, Jha, Somesh, Debray, Saumya, 2007. A semantics-based approach to malware detection. Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, New York, NY, USA, p.377–388, Available from http://doi.acm.org/10.1145/1190216.1190270. [doi:10.1145/1190216.1190270]

Ruan, Guangchen, Tan, Ying, 2010. a three-layer back-propagation neural network for spam detection using artificial immune concentration. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **14**:139-150 (10.1007/s00500-009-0440-2). Available from 10.1007/s00500-009-0440-2

Schultz, Matthew G., Eskin, Eleazar, Zadok, Erez, Stolfo, Salvatore J., 2001. data mining methods for detection of new malicious executables. *Security and Privacy, IEEE Symposium on*, **0**:0038. [doi:10.1109/SECPRI.2001.924286]

, 2009. Pe-miner: Mining structural information to detect malicious executables in realtime.

Tan, Y., Xiao, Z.M., 2007. Clonal particle swarm optimization and its applications. Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, p.2303 -2309. [doi:10.1109/CEC.2007.4424758]

Wang, Jau-Hwang, Deng, P.S., Fan, Yi-Shen, Jaw, Li-Jing, Liu, Yu-Ching, 2003. Virus detection using data mining techinques. Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on, p.71 - 76. [doi:10.1109/CCST.2003.1297538]

Wang, Wei, Zhang, Pengtao, Tan, Ying, He, Xingui, 2009. a hierarchical artificial immune model for virus detection. *Computational Intelligence and Security, International Conference on*, **1**:1-5. [doi:10.1109/CIS.2009.57]