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]*MOE Key Laboratory of Machine Perception, Peking University, Beijing 100871, China*)

([2]*Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China*)

[†]E-mail: weiwang@cis.pku.edu.cn; ytan@pku.edu.cn

Received Dec. 28, 2010; Revision accepted Mar. 6, 2011; Crosschecked May 5, 2011

**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 techniques, the new method does not remember exact file content for virus detection, but uses 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, and then uses an information-theoretic preprocessing 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), radial basis function (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 show that the proposed approach not only has a much faster speed, but also gives around 98% of accuracy.

**Key words:** Local concentration, Artificial immune system, Virus detection
**doi:**10.1631/jzus.C1000445 **Document code:** A **CLC number:** TP301

## 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 the operating system and network (Wang *et al.*, 2009). 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 the organisms' own healthy cells and tissues, and eliminating foreign antigens (Wang *et al.*, 2009). Similarly, the functionality of computer security systems is to recognize and eliminate viruses; thus, the natural immune system has provided with an inspiration to develop this kind of artificial immune based heuristic method for virus detection (Kephart, 1994).

To overcome the disadvantages of the widely used signature-based virus detection method, data

‡ Corresponding author

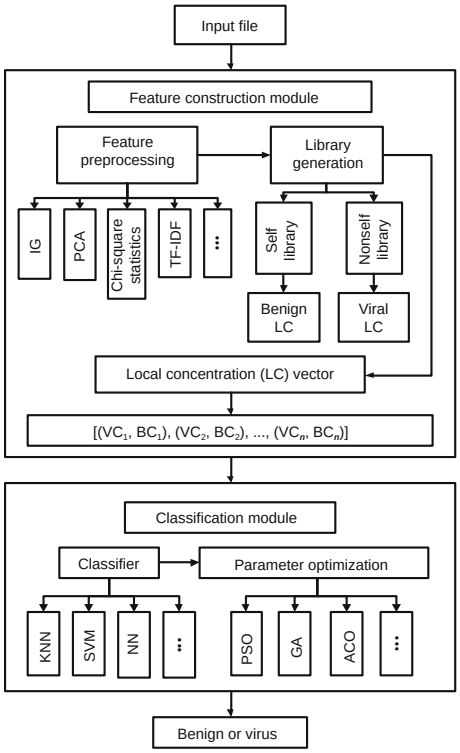
* Project supported by the National Natural Science Foundation of China (Nos. 60673020 and 60875080) and the National High-Tech R & D Program of China (No. 2007AA01Z453)

mining and machine-learning approaches are also proposed for virus detection (Kolter and Maloof, 2006; Christodorescu *et al.*, 2007). Many classification algorithms have been put into practice for solving virus problems combined with an artificial immune system (AIS), including Naïve Bayes, support vector machine (SVM), artificial neural network (ANN), and hybrid approaches (Schultz *et al.*, 2001; Wang *et al.*, 2003; Kolter and Maloof, 2006; Christodorescu *et al.*, 2007).

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 that 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 (Wang *et al.*, 2010). 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 Fig. 1.

Comprehensive experiments were conducted on a public virus data set in the previous works (Chao and Tan, 2009; Wang *et al.*, 2009). Comparisons on performance were made among different classifiers including $K$-nearest neighbor (KNN), radial basis function (RBF) neural networks, and SVM. Experimental results showed that the proposed approach achieves a more than 97% detection rate, and thus outperforms the current approach. The runtime of training and detecting is relatively short. It takes 0.054 s on average to identify a given file.

It is well known that the signature-based virus detection method is incompetent in detecting 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 the signatures' database or matching time of signatures. Different from the existing data mining techniques including the signature-based method and malicious activity detection, the proposed approach neither memorizes specific byte-sequences appearing in the actual file content nor monitors suspicious program behaviors. Our approach is non-signature based and therefore has the potential of detecting previously unknown viruses. Moreover, in most of the current approaches, each selected detector is related to one feature dimension, resulting in a feature of large dimensionality. The proposed model reduces the feature dimensionality and extracts position related information during the process of local concentration extraction. In this way, the model can somewhat overcome 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.




IG: information gain; PCA: principal component analysis; TF-IDF: term frequency–inverse document frequency; VC: 'nonself' local concentration; BC: 'self' local concentration; KNN: $K$-nearest neighbor; SVM: support vector machine; NN: neural network; PSO: particle swarm optimization; GA: genetic algorithm; ACO: ant colony optimization

**Fig. 1 Architecture of our proposed technique**

## 2 Related work

As 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 uses them to detect future instances in similar 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 an existing virus accurately, but it is somewhat limited by the fact that it can identify only a limited amount of generic or extremely broad signatures. It cannot detect some new emerging threats.

Malicious activity detection is another approach to identifying viruses (Ilgun *et al.*, 1995; Hofmeyr *et al.*, 1998). In this approach, the virus detection system monitors the suspicious malicious activities. The file may be further investigated if a malicious behavior is detected. This type of antivirus method does not try to search known similar data, but monitors the malicious activities of the system instead. The malicious activity detection approach can therefore detect new viruses that do not yet appear in any virus dictionaries. Nevertheless, it is a dynamic form of monitoring, and the detection must rely on elements that are 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, have been actively investigated to identify unknown viruses. Since these methods operate for 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.* (2001) and 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 is 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 requirements and has been improved by Kolter and Maloof (2006).

Kolter and Maloof (2006) 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 was 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

Our proposed approach is mainly divided into three parts: (1) generate 'self' and 'nonself' detector libraries from the randomly selected training set; (2) extract the two-element local concentration of each segment in a training sample and connect these local concentrations to construct a feature vector; (3) use three trained classifiers, including KNN, RBF neural networks, and SVM, to detect the testing sample characterized by the ordered concentration vector. The overview of the proposed algorithm is outlined in Algorithm 1.

**Algorithm 1** Immune local concentration based virus detection

```
Generate 'self' and 'nonself' detector libraries by training data
Determine the sizes of the libraries by proportional selection of the potential detectors according to their importance
for each sample in the training set do
    Extract the local concentration vectors of each segment
    Connect these local concentrations to an ordered feature vector as the input of a classifier
end for
Use these local concentration vectors to train the classifier
while algorithm is running do
    if a file is detected then
        Characterize the file by local concentration vectors through trained 'self' and 'nonself' detector libraries
        Use the trained classifier to predict the label of the file
    end if
end while
The dimensionality N of the feature vector is decided by file truncated length
```

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 the 'self' detector library and 'nonself' detector library is as shown in Fig. 2. The concrete step is to divide all detectors into two sets by their tendency values and to calculate the detector importance, with important detectors retained.

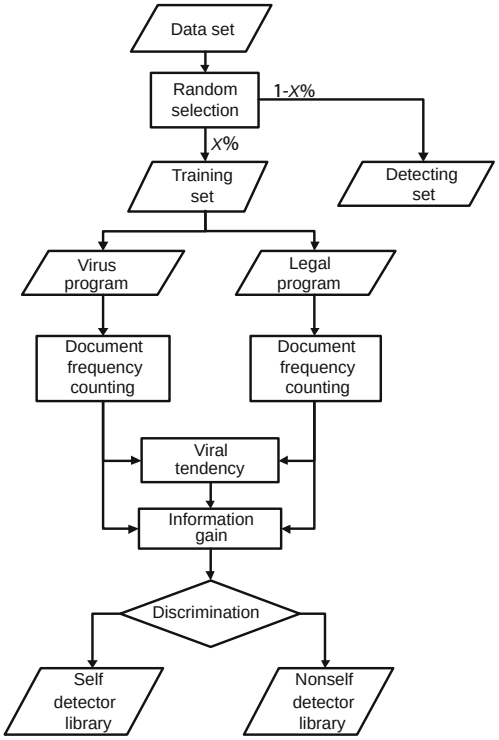



**Fig. 2 Detector library generating process**

The 'self' detector library is composed of detectors most representative of benign files, and the 'nonself' detector library is 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 virus.

As mentioned in Wang *et al.* (2010), 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 a virus from the benign program. The length $L$ is set not too short to discriminate 'self' and 'nonself' or 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 (Fig. 3, the overlap of the 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.

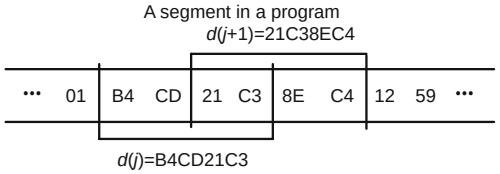



**Fig. 3 Document frequency counting process ($L$=32 bits)**

After counting the document frequency of each fragment, the tendency to be a virus of fragment $X$ is defined as (Wang *et al.*, 2010)

$$T(X) = P(X = 1|C_{\mathrm{v}}) - P(X = 1|C_{\mathrm{s}}), \qquad (1)$$

where $P(X = 1|C_{\mathrm{v}})$ and $P(X = 1|C_{\mathrm{s}})$ mean the document frequencies of fragment $X$ appearing in the virus samples and benign samples of the training set, respectively.

We define the number of virus files as $N_{\mathrm{v}}$, the number of benign files as $N_{\mathrm{s}}$, the number of virus files that contain fragment $X$ as $n_{\mathrm{v}}$, and the number of benign files that contain fragment $X$ as $n_{\mathrm{s}}$. Then,

$$P(X = 1|C_{\mathrm{v}}) = \frac{n_{\mathrm{v}}}{N_{\mathrm{v}}}, \qquad (2)$$

$$P(X = 1|C_{\mathrm{s}}) = \frac{n_{\mathrm{s}}}{N_{\mathrm{s}}}. \qquad (3)$$

If each fragment is extracted to form a dictionary, the size of this dictionary would become very large (Wang *et al.*, 2009). The detectors appearing in most of files are not relevant to the separation of these files because all the classes have instances that contain these detectors. Thus, with the number of detectors growing, the cost of computing would increase, but the effect may not be improved and may be made even worse. We reduce the number of fragments to generate 'self' and 'nonself' detector libraries according to different importance of each detector (Wang *et al.*, 2010). The importance of each detector is calculated based on information gain

(IG). The detectors are sorted based on IG values in descending order; $m\%$ front of them are retained. Besides IG, other detector importance measures including document frequency, term-frequency variance, $\chi^2$ statistic, and principal component analysis (PCA), can also be applied to the model, endowing it with promising development. The preprocessing of statistical and information-theoretic feature generation is therefore complete.

The generation of detector libraries is as described in Algorithm 2, in which $m$ is a parameter to be adjusted, indicating proportional selection of all the fragments. The IG is defined as

$$\mathrm{IG}(X,C)=\sum_{x\in\{0,1\},c\in\{C_\mathrm{v},C_\mathrm{s}\}}\Big(P(X=x\wedge C=c)\cdot\log_2\frac{P(X=x\wedge C=c)}{P(X=x)\cdot P(C=c)}\Big),\tag{4}$$

where $P(X=1)$ means the document frequency of fragment $X$ that appears in the training set, $P(X=0)$ the document frequency of fragment $X$ that does not appear in the training set, $P(C=C_\mathrm{v})$ the document frequency of virus files, $P(C=C_\mathrm{s})$ the document frequency of benign files, $P(X=0|C_\mathrm{v})$ the document frequency of fragment $X$ that does not appear in virus samples of the training set, and $P(X=0|C_\mathrm{s})$ the document frequency of fragment $X$ that does not appear in benign samples of the training set (Wang *et al.*, 2010).

$$P(X=1)=\frac{n_\mathrm{v}+n_\mathrm{s}}{N_\mathrm{v}+N_\mathrm{s}},\tag{5}$$

$$P(X=0)=\frac{N_\mathrm{v}+N_\mathrm{s}-n_\mathrm{v}-n_\mathrm{s}}{N_\mathrm{v}+N_\mathrm{s}},\tag{6}$$

$$P(C=C_\mathrm{v})=\frac{N_\mathrm{v}}{N_\mathrm{v}+N_\mathrm{s}},\tag{7}$$

$$P(C=C_\mathrm{s})=\frac{N_\mathrm{s}}{N_\mathrm{v}+N_\mathrm{s}},\tag{8}$$

$$P(X=0|C_\mathrm{v})=\frac{N_\mathrm{v}-n_\mathrm{v}}{N_\mathrm{v}},\tag{9}$$

$$P(X=0|C_\mathrm{s})=\frac{N_\mathrm{s}-n_\mathrm{s}}{N_\mathrm{s}}.\tag{10}$$

**Algorithm 2** Generation of detector libraries

```
Initialize 'self' and 'nonself' detector libraries as ∅
while algorithm is running do
    for each fragment X in the sample of the training set do
        Calculate the information gain (IG) of fragment X by Eq. (4)
        Sort m% detectors based on IG values in descending order to compose the library // m is a parameter to be adjusted
    end for
    for each fragment X in the library do
        Calculate the tendency of fragment X by Eq. (1)
        if T(X) < 0 then
            Add fragment X into the 'self' detector library
        else
            Add fragment X into the 'nonself' detector library
        end if
    end for
end while
```

## 4 Construction of the 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 'self' local concentration and 'nonself' local concentration (Fig. 4). In every window the local concentration of segment $i$ is defined in Eqs. (11) and (12).

$$\mathrm{VC}_i=\frac{\mathrm{VN}_i\cdot L}{W},\tag{11}$$

$$\mathrm{BC}_i=\frac{\mathrm{BN}_i\cdot L}{W},\tag{12}$$

where $\mathrm{VC}_i$ and $\mathrm{BC}_i$ denote the 'nonself' and 'self' local concentrations, respectively, $\mathrm{VN}_i$ is the number of the detectors appearing in both the detecting segment of the file and the 'nonself' detector library, and $\mathrm{BN}_i$ is the number of the detectors appearing in both the detecting segment of the file and the 'self' detector library.

After 'self' and 'nonself' local concentrations are constructed in each window, these two-element local concentrations of the program are connected to form a feature vector $[(\mathrm{VC}_1, \mathrm{BC}_1), (\mathrm{VC}_2, \mathrm{BC}_2), \cdots, (\mathrm{VC}_n, \mathrm{BC}_n)]$ (Fig. 5).

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

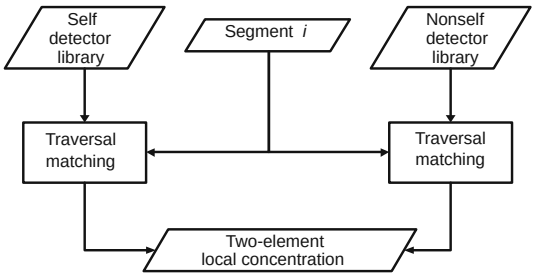



**Fig. 4 Local concentration construction**

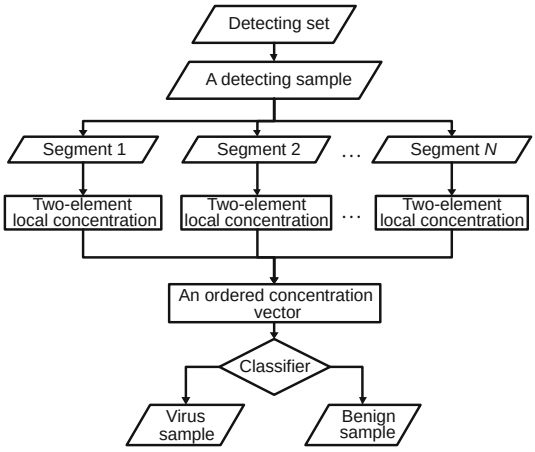



**Fig. 5 Feature vector construction**

## 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 predefined 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, etc. can be used as the distance metrics according to different situations.

**Algorithm 3** Feature construction

1: **for** a program to be detected **do**
2: Truncate front $N \times W$ bits of the file and discard rear dimensionality of the file
3: **for** each segment inside $W$-bit jumping windows **do**
4: Traverse the segment $i$ using an $L$-bit sliding window with $L/2$ bits overlap
5: Use the global concentration method as given in our previous work (Wang *et al.*, 2010)
6: Calculate the 'nonself' local concentration $\mathrm{VN}_i$
7: Calculate the 'self' local concentration $\mathrm{BN}_i$
8: Label the feature vector of segment $i$ with ($\mathrm{VN}_i$, $\mathrm{BN}_i$)
9: **end for**
10: Connect these ordered two-element local concentrations to construct a feature vector
11: **end for**

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, $K$ 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

An SVM is a supervised learning algorithm. The algorithm constructs a hyperplane or a 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 classifier. 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 the optimal hyperplane in the feature space and hence the performance of classification.

### 5.3 RBF neural network

Neural network is an adaptive system whose structure changes based on the 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 RBF as the activation function. Like the architecture of the standard feed forward back-propagation 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 study, the 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 is more powerful in a multi-dimensional space. It tends to have several times more neurons than a comparable feedforward network in the hidden layer, and each neuron responds only 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 can easily find the minimum. Moreover, the 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 RBF. A larger spread leads to a smoother RBF and more neurons responding to an input vector. A smaller spread leads to a steeper RBF, 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 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

Experiments were conducted on a public virus data set (Chao and Tan, 2009; Wang *et al.*, 2009). The 'cilpku08' data set was obtained from a famous virus website VX Heavens (http://vx.netlux.org/) and from Computer-Forensic Experts of Anti-Virus Group in Peking University, which can be obtained from http://www.cil.pku.edu.cn/resources/. The folder includes 3547 malicious executables classified into 685 families based on their properties, comprising 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 were obtained from all folders of machines running the Windows 2000 and XP operating systems. This data set was divided into three subsets (Chao and Tan, 2009; Wang *et al.*, 2009). 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 much smaller than and is covered by the detecting set, so that the expansibility and comprehensive ability can be tested.

The test platform setting for experiments is shown in Table 1.

**Table 1 The test platform**

| | Description |
|---|---|
| Operating system | Windows XP |
| Computer hardware | CPU: Pentium IV 1.5 GHz; RAM: 512 MB |
| Programming language | C & Matlab languages |
| Compiling environment | Microsoft Visual C++ 6.0 & Matlab R2007a |

### 6.1 Experiments for different window sizes and numbers of windows

In this part, different window size $W$ and number of windows $N$, corresponding to the dimensionality of the feature vector, were 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 of 100 and $N$ ranges from 20 to 60 with a step size of 10. The average results of 10 experiments with different partitions of the second data set were used to measure the performance.

Figs. 6–8 show that when $W$=400 and $N$=50 the results are considerably stable and quite good on the data set. Thus, these two parameters were fixed in the following experiments.

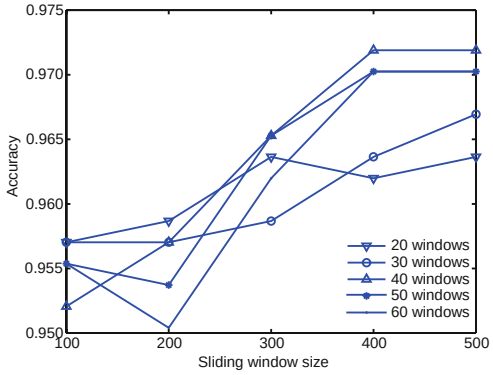



**Fig. 6 Accuracy with different window sizes and numbers of windows on the second data set by *K*-nearest neighbor (KNN)**

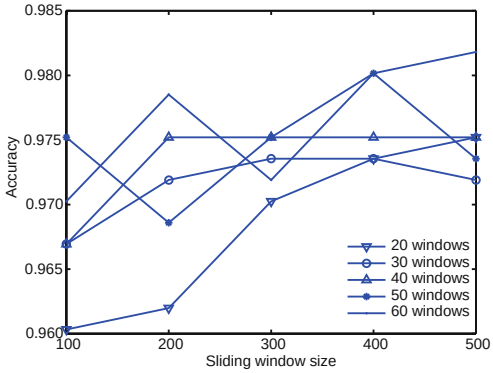



**Fig. 7 Accuracy with different window sizes and numbers of windows on the second data set by support vector machine (SVM)**

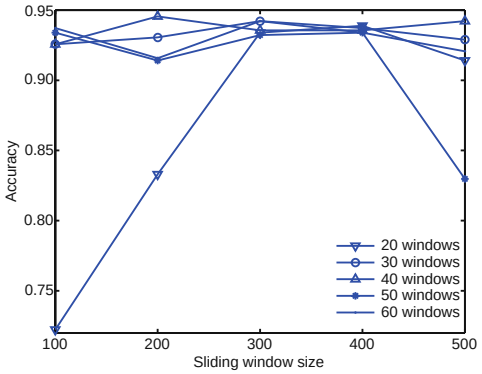



**Fig. 8 Accuracy with different window sizes and numbers of windows on the second data set by radial basis function neural network (RBF NN)**

### 6.2 Experiments for different proportional selections of all the fragments

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

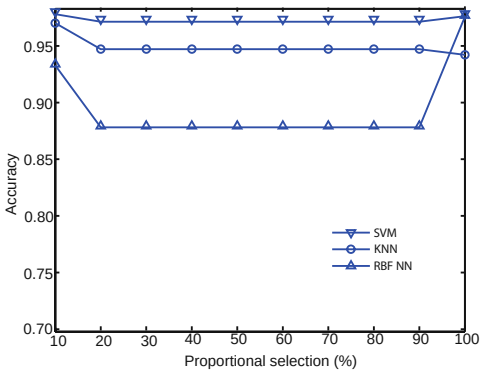



**Fig. 9 Accuracy with different proportional selections of the fragments on the second data set for the three data mining methods (window size: 400; window number: 50)**

When $m$=10%, the detecting rate has the best performance; i.e., the size of the detector dictionary is the 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, 32, or 64 bits, $L$ is not necessarily larger than 64 to contain at least one entire instruction. The length of the detector was taken as 32 or 64 in this work 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 the sliding window is $L/2$ bits. The accuracy rates using the SVM classifier with different lengths of the detector are shown in Table 2.

### 6.4 Contrast experiments

To assess the performance and 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 a Windows operating system. The same partitions were made using the same data. Tests 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, respectively. Tests 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, respectively. Tests 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, respectively.

As shown in Figs. 10 and 11, our proposed method outclassed the hierarchical AIS (H-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 was growing. The runtime performance of our method was also better than that of the H-AIS method. In Fig. 12, the training time of the new method varies linearly with the number of files, unlike the H-AIS method whose training time grew exponentially with the file number. Furthermore, the runtimes of the new method (several minutes) and the H-AIS method (several hours) are not of the same order of magnitude.

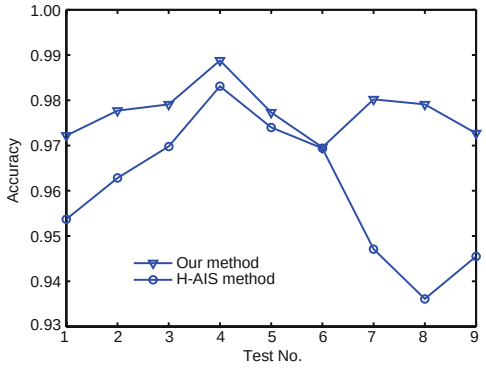



**Fig. 10 Accuracy rate on the detecting sets of contrast experiments (proportional selection: 10%; window size: 400; window number: 50)**

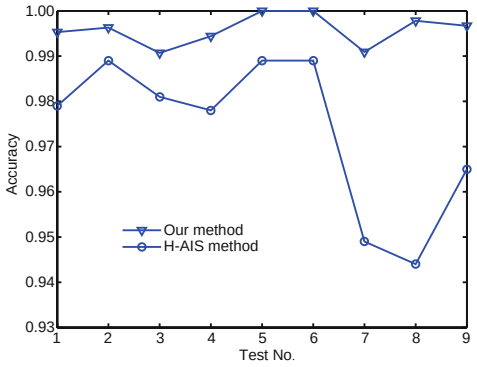



**Fig. 11 Accuracy rate on the training sets of contrast experiments (proportional selection: 10%; window size: 400; window number: 50)**

**Table 2 Average accuracy rate by SVM when the detector length $L$=64 or 32**

| Test No. | 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 |
| 1 | 100.00 | 100.00 | 100.00 | 97.22 | 96.08 | 98.25 | 99.53 | 99.51 | 99.56 | 95.37 | 98.04 | 92.98 |
| 2 | 100.00 | 100.00 | 100.00 | 97.40 | 95.28 | 99.30 | 99.63 | 99.21 | 100.00 | 98.51 | 96.85 | 100.00 |
| 3 | 100.00 | 100.00 | 100.00 | 94.19 | 87.68 | 100.00 | 99.07 | 98.04 | 100.00 | 96.74 | 95.57 | 97.80 |
| 4 | 100.00 | 100.00 | 100.00 | 97.75 | 96.43 | 98.94 | 99.44 | 98.82 | 100.00 | 96.07 | 92.86 | 98.94 |
| 5 | 100.00 | 100.00 | 100.00 | 97.03 | 94.49 | 99.30 | 100.00 | 100.00 | 100.00 | 97.03 | 98.43 | 95.77 |
| 6 | 100.00 | 100.00 | 100.00 | 95.56 | 90.59 | 100.00 | 100.00 | 100.00 | 100.00 | 95.00 | 90.59 | 98.95 |
| 7 | 99.92 | 100.00 | 99.84 | 98.02 | 97.67 | 98.36 | 99.09 | 99.50 | 98.69 | 98.02 | 97.67 | 98.36 |
| 8 | 100.00 | 100.00 | 100.00 | 96.04 | 94.00 | 98.03 | 99.78 | 99.56 | 100.00 | 97.91 | 97.33 | 98.47 |
| 9 | 100.00 | 100.00 | 100.00 | 95.12 | 91.83 | 98.36 | 99.67 | 99.33 | 100.00 | 97.27 | 95.83 | 98.69 |

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 Fig. 13.

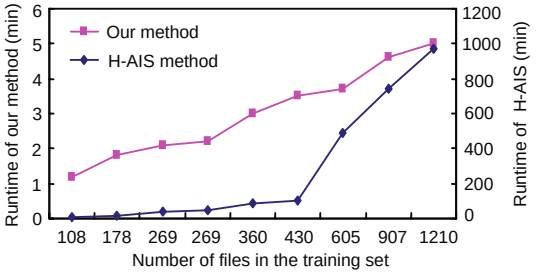



**Fig. 12 Training runtime of two methods**

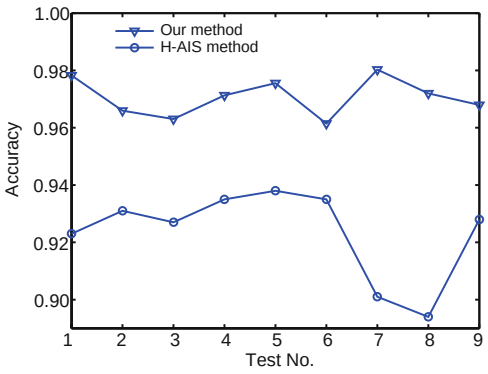



**Fig. 13 Accuracy rate on the detecting sets of expanded contrast experiments by SVM (proportional selection: 10%; window size: 400; window number: 50)**

### 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, and the window number $N$ setting, which in turn determine the feature vector, are here an optimization problem.

The vector that needs to be optimized, $\boldsymbol{P}^* = [m, W, N, P_1^*, P_2^*, \cdots, P_n^*]$, 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 (Wang *et al.*, 2010).

When $m$ is set to different values, different detector libraries are obtained. A unique feature vector can be constructed with different $W$ and $N$, for a file to be characterized, whose '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 example, parameters associated with KNN include a number of nearest neighbors and the distance measures. SVM-related parameters that determine the position of the optimal hyperplane in the 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 that makes the accuracy of classification maximum. The cost function $\mathrm{CF}(\boldsymbol{P})$ can be defined as

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

where $\mathrm{Err}(\boldsymbol{P})$ is the classification error on the training set.

The input vector $\boldsymbol{P}$ is composed of two parts: LC feature vector determinants $m$, $W$, $N$ and the classifier-related parameters $P_1^*, P_2^*, \cdots, P_m^*$. It is to find a $\boldsymbol{P}^*$ such that

$$\begin{aligned}\mathrm{CF}(\boldsymbol{P}^*) &= \mathrm{Err}(\boldsymbol{P}^*)\\ &= \min_{\{m,W,N,P_1^*,P_2^*,\cdots,P_m^*\}} \mathrm{Err}(\boldsymbol{P}).\end{aligned} \tag{14}$$

Several robust optimization approaches can be used to optimize the input vector, such as particle swarm optimization (PSO) and the genetic algorithm (GA). Here, we use a clonal PSO (CPSO) (Fig. 14) to design the LC feature vector and the corresponding classifier. For the detailed optimization process, refer to Tan and Xiao (2007).

The selection of the LC feature vector determinants $m$, $W$, $N$ and the classifier-related parameters, $P_1^*$, $P_2^*$, $\cdots$, $P_m^*$, is a dynamic optimization process (which is the same as the parameter selection in Wang *et al.* (2010)). Parameters associated with KNN include the number of nearest neighbors $K$ and the distance measures. $K$ is optimized in the integer number interval [1, 20], and the distance measures are chosen among 'Euclidean', 'cityblock', 'cosine', and 'correlation'. For SVM, the cost parameter $C$ is optimized in real number interval [1, 200]. For the RBF neural network, the spread $\sigma$ in real number interval [1, 5] is optimized. $m$ is optimized in

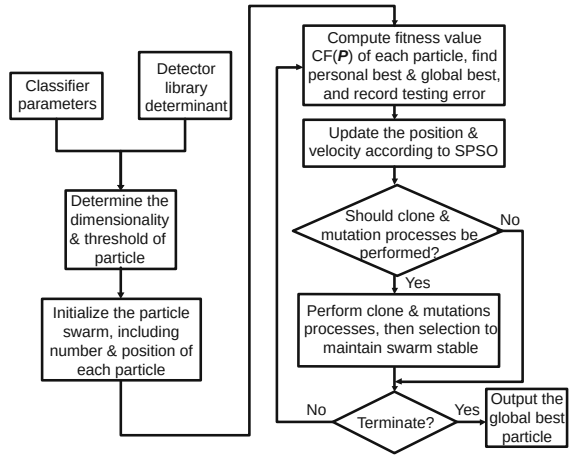



**Fig. 14 Clonal particle swarm optimization (CPSO) based classification process**

the integer number interval [5, 100], and $W$ and $N$ range in [100, 600] and [10, 100], respectively. The maximum number of generations is set to be 200 as the stop criterion, and the number of particles in a swarm is 20.

The randomness of CPSO leads to a slight variation in the performance and parameters obtained. Therefore, the results of nine independent classes of experiments on the expended third data set were used to evaluate tests. This design is more reasonable. The average performances of empirical and optimized classification designs are reported in Tables 3 and 4.

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

| Test No. | Average accuracy rate (%) | | False positive rate* (%) | |
|---|---|---|---|---|
| | Optimized | Empirical | Optimized | Empirical |
| 1 | 97.83 | 97.83 | 4.37 | 4.37 |
| 2 | 98.51 | 96.59 | 0.00 | 2.95 |
| 3 | 96.74 | 96.30 | 2.20 | 3.61 |
| 4 | 97.75 | 97.13 | 1.06 | 3.17 |
| 5 | 97.56 | 97.56 | 3.93 | 3.93 |
| 6 | 96.15 | 96.15 | 3.17 | 3.17 |
| 7 | 98.03 | 98.03 | 1.64 | 1.64 |
| 8 | 97.91 | 97.20 | 1.53 | 1.09 |
| 9 | 97.27 | 96.80 | 1.31 | 1.20 |
| Average | 97.53 | 97.06 | 2.14 | 2.79 |

* 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.

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

| Test No. | Average accuracy rate (%) | | False positive rate* (%) | |
|---|---|---|---|---|
| | Optimized | Empirical | Optimized | Empirical |
| 1 | 97.78 | 97.78 | 4.04 | 4.04 |
| 2 | 97.77 | 96.08 | 0.70 | 2.84 |
| 3 | 97.91 | 96.23 | 0.00 | 3.06 |
| 4 | 98.88 | 97.69 | 1.06 | 4.04 |
| 5 | 96.65 | 96.12 | 0.00 | 2.73 |
| 6 | 95.94 | 95.94 | 3.61 | 3.61 |
| 7 | 97.02 | 96.68 | 0.00 | 0.44 |
| 8 | 97.47 | 96.30 | 1.31 | 1.20 |
| 9 | 96.94 | 95.47 | 1.15 | 0.87 |
| Average | 97.37 | 96.48 | 1.32 | 2.54 |

* The rate of legal files mistakenly classified as malicious executables

## 7 Conclusions

In this paper we have proposed a non-signature based approach that analyzes the byte-level file content. In contrast to 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 a 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, R., Tan, Y., 2009. A Virus Detection System Based on Artificial Immune System. Int. Conf. on Computational Intelligence and Security, **1**:6-10. [doi:10.1109/CIS.2009.106]

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

Christodorescu, M., Jha, S., Kruegel, C., 2007. Mining Specifications of Malicious Behavior. Proc. 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering, p.5-14. [doi:10.1145/1287624.1287628]

Egele, M., 2008. Behavior-Based Spyware Detection. VDM Verlag, Saarbrucken, Germany.

Henchiri, O., Japkowicz, N., 2006. A Feature Selection and Evaluation Scheme for Computer Virus Detection. Proc. 6th Int. Conf. on Data Mining, p.891-895. [doi:10.1109/ICDM.2006.4]

Hofmeyr, S.A., Forrest, S., Somayaji, A., 1998. Intrusion detection using sequences of system calls. *J. Comput. Secur.*, **6**:151-180.

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

Jacob, G., Debar, H., Filiol, E., 2008. Behavioral detection of malware: from a survey towards an established taxonomy. *J. Comput. Virol.*, **4**(3):251-266. [doi:10.1007/s11416-008-0086-0]

Kephart, J.O., 1994. A Biologically Inspired Immune System for Computers. Proc. 4th Int. Workshop on Synthesis and Simulatoin of Living Systems, p.130-139.

Kirda, E., Kruegel, C., Banks, G., Vigna, G., Kemmerer, R.A., 2006. Behavior-Based Spyware Detection. Proc. 15th Conf. on USENIX Security Symp., p.1-16.

Kolter, J.Z., Maloof, M.A., 2006. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.*, **7**:2721-2744.

Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J., 2001. Data Mining Methods for Detection of New Malicious Executables. Proc. IEEE Symp. on Security and Privacy, p.38-49. [doi:10.1109/SECPRI.2001.924286]

Tan, Y., Xiao, Z.M., 2007. Clonal Particle Swarm Optimization and Its Applications. IEEE Congress on Evolutionary Computation, p.2303-2309. [doi:10.1109/CEC.2007.4424758]

Wang, J., Deng, P.S., Fan, Y., Jaw, L., Liu, Y., 2003. Virus Detection Using Data Mining Techinques. Proc. IEEE 37th Annual Int. Carnahan Conf. on Security Technology, p.71-76. [doi:10.1109/CCST.2003.1297538]

Wang, W., Zhang, P.T., Tan, Y., He, X.G., 2009. A Hierarchical Artificial Immune Model for Virus Detection. Int. Conf. on Computational Intelligence and Security, **1**:1-5. [doi:10.1109/CIS.2009.57]

Wang, W., Zhang, P.T., Tan, Y., 2010. An immune concentration based virus detection approach using particle swarm optimization. *LNCS*, **6145**:347-354. [doi:10.1007/978-3-642-13495-1_43]

**Introducing editorial board member:** Professor Xin-gui He is an editorial board member of *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)*. He is a PhD supervisor of computer science at Peking University. He was the dean of School of Electronics Engineering and Computer Science from 2002 to 2006. Professor He received his bachelor degree from Peking University in 1960, and later as a graduate student majored in approximation theory in the same university. He has been a member of the Chinese Academy of Engineering since 2001. His main research interests include fuzzy logic, artificial neural network, evolutionary computation, and database theory.