J Comput Virol (2011) 7:267–277
DOI 10.1007/s11416-011-0154-8

ORIGINAL PAPER

Run-time malware detection based on positive selection

Zhang Fuyong · Qi Deyu

Received: 16 November 2010 / Accepted: 13 July 2011 / Published online: 28 July 2011
© Springer-Verlag France 2011

Abstract This paper presents a supervised methodology that detects malware based on positive selection. Malware detection is a challenging problem due to the rapid growth of the number of malware and increasing complexity. Run-time monitoring of program execution behavior is widely used to discriminate between benign and malicious executables due to its effectiveness and robustness. This paper proposes a novel classification algorithm based on the idea of positive selection, which is one of the important algorithms in Artificial Immune Systems (AIS), inspired by positive selection of T-cells. The proposed algorithm is applied to learn and classify program behavior based on I/O Request Packets (IRP). In our experiments, the proposed algorithm outperforms ANSC, Naïve Bayes, Bayesian Networks, Support Vector Machine, and C4.5 Decision Tree. This algorithm can also be used in general purpose classification problems not just two-class but multi-class problems.

1 Introduction

According to the reports published by Symantec in April 2010, the number of malware is rapidly increasing. In 2008 the number of exiting malware increased up to 1691 thousand and in 2009 grows up to 2895 thousand [1]. The rapid growth of the number of malware has made manual methods of disassembly or reverse engineering unaffordable [2]. More resilient and effective methods must be applied to combat malware.

Z. Fuyong (✉) · Q. Deyu
Research Institute of Computer Systems, South China University of Technology, Guangzhou 510006, China
e-mail: z.fuyong@mail.scut.edu.cn; fuyong1681@163.com

Q. Deyu
e-mail: qideyu@mail.scut.edu.cn

Run-time malware detection strategies have attracted extensive attention due to its effectiveness and robustness. Forrest et al. [3] leverage system calls to discriminate between benign and malicious Unix processes. Hofmeyr et al. [4] build normal behavior of Unix processes in terms of short sequences of system calls. The Hamming distance is used to determine how closely a system call sequence resembles another. A threshold must be set to determine whether a process is anomalous. Typically, processes showing large Hamming distance values are anomalous. Wepsi et al. [5] propose an improved version with variable length system call sequences. A detection method based on the frequency of system calls has been proposed by Sato et al. [6]. Manzoor et al. [7] collect some Windows malicious executables from VX Heavens [8] and their API call sequences are monitored by API Monitor [9]. The DCA (Dendritic Cell Algorithm) [10–12] is applied for detection. Later, Ahmed et al. [13] use statistical features which extracted from both spatial (arguments) and temporal (sequences) information available in Windows API calls for malware detection. All these methods use system calls or API calls to monitor program behavior. However, the system call or API call sequences can be manipulated by a crafty attacker to circumvent detection [14–16].

Seifert et al. [17] compared three popular event-based techniques that can monitor program behavior: user mode API hooking, kernel mode API hooking, and kernel mode callbacks. Applications that directly call the kernel and avoid using the Win32 API cannot be monitored by user mode API hooking. Kernel mode API hooking injects code into the kernel itself providing a system wide view of program behavior that is more difficult to circumvent [18]. Modifications of the kernel in this manner are now being discouraged by the vendor, as they can cause other programs to crash or perform unexpectedly. Kernel mode API hooking is not

portable across different versions of the operating system as hooked function calls are not a supported interface and therefore are changing with each version. Instead of patching the kernel with kernel mode API hooking, Microsoft encourages the usage of callback functions [19]. Callback functions are publicly supported interfaces on the kernel mode that notify an application about state changes on the system. These callbacks are designed with reliability and long-term supportability in mind allowing a monitoring application to run on various versions of the Microsoft Windows operation system without modification [17]. So kernel mode callbacks is the best way to monitor program behavior, and I/O Request Packets (IRP) are used to analyze program behavior, MBMAS [20] is used for capturing IRPs in this paper.

Artificial Immune Systems (AIS) are defined as intelligent computational systems inspired by Human Immune System (HIS), which are applied to anomaly detection [3,21–23], optimization [24,25], clustering and classification [26–30], and so on. The most widely used theories in AIS are self/nonself theory and clonal selection theory. The representative algorithms are the negative selection algorithm (NSA) presented by Forrest et al. [21], which is inspired by the process of self-tolerance of B-cells, and CLONALG [24,31], which is inspired by clonal selection theory and consists of mutation and selection processes. NSA is appropriate for anomaly and malware detection problems. In original NSA, candidate detectors are generated in a random manner, and then exposed to a negative censoring mechanism. Only the qualified detectors that do not match any self-sample are inserted to the detector set. Unfortunately, these randomly generated detectors have three problems. Firstly, many of these randomly generated detectors are useless and will be discarded. Secondly, many detectors generated in a random manner are similar. In other words, some nonself data are detected by two or more detectors. Thirdly, these randomly generated detectors cannot be guaranteed to cover the nonself-space [44]. Particle swarm optimization was proposed to optimize detectors of negative selection algorithm [45]. This method uses a multi-phase particle swarm optimization and anti-collision technique. Each sub-swarm represents a detector group, which consists of a given number of detectors. But their methods use fixed radius for each detector. The real valued negative selection was proposed in order to cover the abnormal space [46]. They introduced a new scheme of detector generation and matching mechanism for negative selection algorithm with variable properties. The negative selection was applied to multi-class classification problems [29]. Detectors are generated for each self and nonself data set using clonal selection principles and a new data reduction technique is proposed to reduce the noise effect. All these algorithms have two problems. Firstly, in many cases self-space is small and nonself-space is large. In particular, nonself-space is larger than self-space for multi-class classification problems. Large numbers of detectors are needed to cover nonself-space. Secondly, in these algorithms, when a test sample cannot be recognized by any detector, each detector radius is enlarged by a fixed multiplier until this sample is recognized by at least one detector. The problem is that more than one enlarged detectors covers the self-space. In addition, many detectors overlap the others. Therefore, one test data can be recognized by multiple detectors.

In HIS, only T-cells capable of binding to Self-MHC (Major Histocompatibility Complex) molecules can survive [34]. This is called positive selection [32,33]. It is also an important mechanism in AIS. Positive selection algorithm (PSA), inspired by the positive selection, belongs to self/nonself theory and is the opposite of NSA. In PSA, the detectors that match any self-sample are inserted into the detector set. In other words, the detectors only need to cover self-space. In multi-class classification problems, self-space is smaller than nonself-space. So, fewer detectors are needed in PSA.

In this paper, we present a novel classification algorithm, positive selection classification algorithm (PSCA), based on the idea of positive selection. PSCA overcomes the two above mentioned problems. Firstly, only self-space are needed to be covered. Secondly, k-nearest neighbor algorithm is used to solve the hole problem. So not all self-space are needed to be covered. In other words, fewer detectors are needed. We also propose an initial algorithm for classifier generation based on the maximum distance to cover more self-space with fewer classifiers, and clonal selection algorithm is used to search approximate optimal classifiers. For general purpose, we expand PSCA to handle multi-class problems. PSCA turns multi-class classification problem into two-class classification problem: self and nonself. Each time one class is selected as self-class and the others are nonself-class. The classifiers which only can recognize self-class data are selected by positive selection. The procedure is iterated until classifiers are generated for all classes. At classification stage, these classifiers are used to recognize self-class data. In our experiments, PSCA outperforms artificial negative selection classifier (ANSC) [29], Naïve Bayes (NB), Bayesian Networks (BN), Support Vector Machine (SVM), Decision Tree (DT), NB with Boosting (BNB), BN with Boosting (BBN), SVM with Boosting (BSVM), and DT with Boosting (BDT).

The contributions are: (1) Kernel mode callbacks technique, which is more effective and robust than API hooking, is used to monitor program behavior and IRPs are used for run-time malware detection; (2) PSCA is proposed for effective classification.

This paper is organized as follows: Sect. 2 introduces the positive selection algorithm and clonal selection algorithm in AIS. Section 3 describes PSCA in detail. The experimental results and discussion are given in Sect. 4. Section 5 concludes this paper.

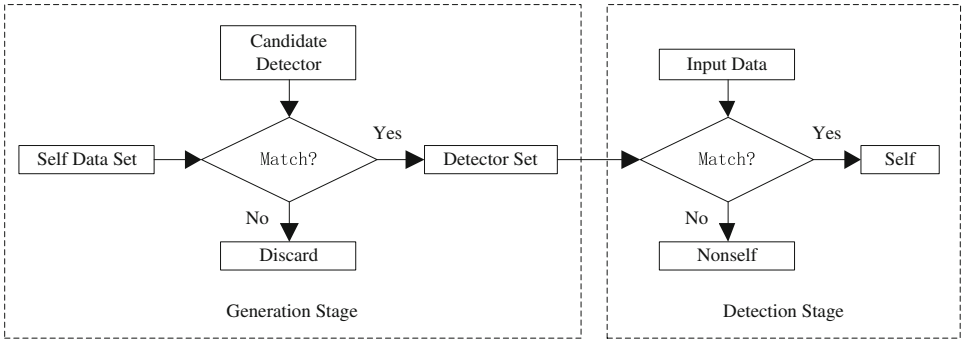

Fig. 1 Positive selection algorithm

2 Artificial immune system

2.1 Positive selection algorithm

Positive selection algorithm [33] is inspired by positive selection process of T-cells. In the selection process, only T-cells able to recognize self-molecules can be used in immune system. Seiden and Celada [32] proposed a model for simulating cognate recognition and response in the immune system, in which the positive selection algorithm is applied.

The concept of the positive selection algorithm is shown in Fig. 1. At the generation stage, the candidate detector is produced randomly and tested for the ability to recognize self-samples. A detector which is able to recognize any self-sample is added to the detector set. Only those that cannot recognize any self-sample are removed. At the detection stage, if one input data is recognized by any detector, result is self-class; otherwise, result is nonself-class.

2.2 Clonal selection algorithm

When an immune system is exposed to an antigen, the antibodies which spread on surfaces of B lymphocytes are produced. Each B-cell is specific to a given antigen. Antibodies are able to recognize certain type of antigens. If a new antigen enters into the body, the immune system clones the most stimulated lymphocytes. Clonal selection has a mutation operator. The mutation rate of an individual is inversely proportional to its affinity by means of different mutation variations. In other words, the better affinity the antibody has, the less it may be exposed to mutation. A general scheme for clonal selection is CLONALG [24,31]. It utilizes clonal selection and affinity maturation. The main steps of this algorithm are described in the following.

1. The algorithm starts with an initial set of population P_r of B-cells and an empty memory set M.
2. The selection process then selects n best cells from P_r to generate a new population P_n according to the affinity principle.
3. The clonal process reproduces a population of clones C from the population of P_n cells. This step produces more offspring for higher affinity cells.
4. The maturation process mutates the cells to create the population C^*.
5. The reselection process reselects the improved cells from C^* and updates the memory set M.
6. The diversity introduction process replaces d cells with new ones N_d.

This algorithm is one of the most classic clone selection algorithms, which is applied to pattern recognition [35], anomaly detection [36], and so on. The clone selection method used in this paper is based on this algorithm.

3 Positive selection classification algorithm

Positive Selection Classification Algorithm (PSCA) is a general classification algorithm. PSCA classifies unknown data only use classifiers that are able to recognize self-class data. In order to obtain effective classifiers, PSCA uses positive selection algorithm and clonal selection algorithm in AIS. The following notations are applied in the PSCA.

- D: the training data set.
- d: a training data, $d \in D$.
- n_f: the number of features in the data set.
- n_c: the number of classes in the data set.
- $d.f$: the feature vector of the training data d.
- $d.flg$: the recognized flag of the training data d.
- $d.f_i$: the value of the ith feature in $d.f$.
- D_i: the training data set of the ith class, $D_i \subseteq D$, $D = \{D_1 \cup D_2 \cup \ldots \cup D_n\}$.
- Let one class (i) be the self-class, and the set of the remaining classes $(1, \ldots, i-1, i+1, \ldots, n_c)$ be the nonself-class.
- S: the self-class data set, $S = D_i$.
- $|S|$: the number of self-class data set.
- s: a self-class data, $s \in S$.

- N: the nonself-class data set, $N = \{D_1 \cup \ldots \cup D_{i-1} \cup D_{i+1} \cup \ldots \cup D_n\}$.
- n: a nonself-class data, $n \in N$.
- C: the classifier set.
- c: a classifier, $c \in C$.
- $c.f$: the feature vector of classifier c.
- $c.r$: the recognition radius of classifier c. If the distance between the classifier c and the training data d is less than the recognition radius of c, then c can recognize d. The distance is dependent on the type of data, for example, the Euclidean distance for real-valued data or the Hamming distance for bitstrings.
- $c.sl$: the stimulation level of classifier c. The stimulation level is the number of self-class data recognized by the new classifier but not recognized by any old classifier.
- $c.ssl$: the substimulation level of classifier c. The substimulation level is the number of self-class data recognized by the new classifier and old classifiers.
- C_i: the classifier set of the ith class, $C_i \subseteq C$, $C = \{C_1 \cup C_2 \cup \ldots \cup C_n\}$.
- N_g: number of iterations.
- N_c: number of clone selections.
- P: mutation probability.

3.1 Learning stage

The purpose of the learning stage is to produce classifiers for each class. This process turns multi-class classification problem into two-class classification problem: self and nonself. The functions used at learning stage are described as follows.

- $minDist\_s(s, N)$: the minimum distance between s and each n in N.
- $minDist\_c(c, N)$: the minimum distance between c and each n in N.
- $distOrder(S, N)$: the descending order of all $minDist\_s(s, N)$, $s \in S$.
- $distant[|S|]$: the self-class data set obtained by $distOrder(S, N)$.
- $getClassifier(s, S, N)$: generate a classifier.
- $setRadius(c, N)$: set the recognition radius of c.
- $setStimulation(c, S)$: set the stimulation and substimulation level of c.
- $mutate(c)$: mutation of c.
- $clone(c\_clone, c\_best)$: clone selection.
- $recognize(c, s)$: return $true$ if c recognizes s; return $false$ otherwise.

The classifiers, which are generated by PSCA, only recognize self-class data. The initial algorithm for classifier generation based on the maximum distance is proposed to generate effective initial classifiers. The maximum distance is

$$\max_{s \in S} minDist\_s(s, N) \tag{1}$$

This method sorts the minimum distances between each self-class data and all nonself-class data in descending order. The first data in the sorted list which is not recognized by any existing classifier is used as the feature value for the initial classifier. Algorithm 1 shows the learning stage of PSCA.

Algorithm 1 Learning stage of PSCA

```
for each D_i(S) in D do
  C_i ← ∅;
  distOrder(S, N);
  for each s in S do
    s.flg ← false;
  end for
  for i ← 0 to |S| do
    if distant[i].flg eq false then
      c ← getClassifier(distant[i], S, N);
      C_i ← C_i ∪ c;
    end if
    for j ← 0 to |S| do
      if recognize(c, distant[j]) then
        distant[j].flg ← true;
      end if
    end for
  end for
end for
```

In Algorithm 1, the function $getClassifier(s, S, N)$ is used to set the feature vectors, recognition radius, stimulation level, and substimulation level. The optimal classifier is obtained by clonal selection. Details of $getClassifier(s, S, N)$ are described in Algorithm 2. The recognition radius set function is determined by:

$$c.r = minDist\_c(c, N) \times \alpha \tag{2}$$

where α is the control parameter, $\alpha \in [0, 1]$. Equation 2 ensures that the classifier does not recognize any nonself-class data, and reduces the probability of misclassification.

Algorithm 2 $getClassifier(s, S, N)$

```
c.f ← s.f;
c.r ← setRadius(c, N);
c ← setStimulation(c, S);
c_best ← c;
for i ← 0 to N_g do
  for j ← 0 to N_c do
    c_clone ← mutate(c);
    c_clone.r ← setRadius(c_clone, N);
    c_clone ← setStimulation(c, S);
    c_best ← clone(c_clone, c_best);
  end for
  c ← c_best;
end for
return c;
```

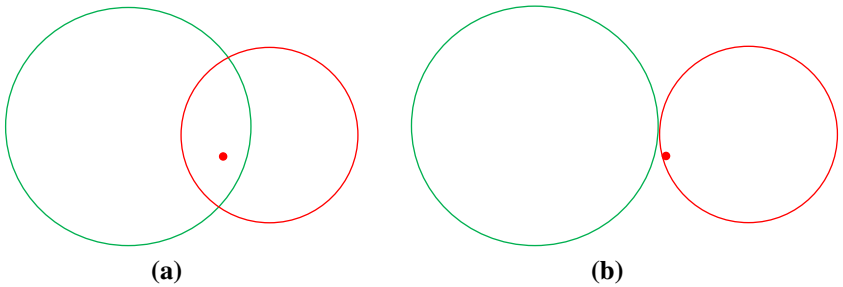

Fig. 2 **a** Overlap. **b** The solution to overlap

Algorithm 3 $setStimulation(c, S)$

```
c.sl ← 0;
c.ssl ← 0;
for each s in S do
  if recognize(c, s) eq true then
    if s.flg then
      c.ssl ← c.ssl + 1;
    else
      c.sl ← c.sl + 1;
    end if
  end if
end for
return c;
```

Algorithm 4 $mutate(c)$

```
flg ← false;
while flg eq false do
  for each c.f_i in c.f do
    if random(0, 1) < P then
      if c.f_i eq 0 then
        c.f_i ← 1;
      else
        c.f_i ← 0;
      end if
      flg ← true;
    end if
  end for
end while
return c;
```

Algorithm 5 $clone(c\_clone, c\_best)$

```
if c_clone.sl > c_best.sl then
  return c_clone;
else
  if (c_clone.sl eq c_best.sl)&&(c_clone.ssl > c_best.ssl) then
    return c_clone;
  else
    return c_best;
  end if
end if
```

Algorithm 3 describes the function $setStimulation(c, S)$. Algorithm 4 describes the mutation process. Each feature of a classifier is adjusted according to the mutation probability P. The mutation process is used to search for the optimal classifier in the whole space. The clonal selection process is shown in Algorithm 5, which is used to determine whether or not the mutated classifier is better than the original one. The judgement rules are:

1 If the stimulation level increased after mutation, the mutated classifier is better;
2. If the stimulation level is equal to the original stimulation level after mutation, and the substimulation level increased, then the mutated classifier is better;
3. Otherwise, the original classifier is better.

3.2 Classification stage

After the learning stage, all classifiers are available to classify the unknown data (s). The radius is a threshold used for classification, opposed to the usual classification approach where the minimal distance between several centers is used. If the distance between classifier (i) and s less than the radius of i, i can recognize s. If only one kind of classifiers i recognize s, the class of s is determined as i. This is the normal state, however there are two other states: (1) overlap: the unknown data is recognized by more than two kinds of classifiers (Fig. 2a). (2) hole: no classifiers can recognize the unknown data (Fig. 3a). The solutions to these two states are describes as follows.

1. Overlap: Fig. 2a describes the state that the unknown data is recognized by two kinds of classifiers. The red point in Fig. 2 represents the unknown data. In this situation, the two classifiers are temporarily moved away along the direction of the center connection. The moving distance is $(c_1 + c_2 - dist(c_1, c_2))/2$. c_1 and c_2 represent the two classifiers, and $dist(c_1, c_2)$ is the distance of the two classifiers. Figure 2b is the state after the movement. The two circles are tangent with each other. After the movement, if the unknown data is recognized by only one kind of classifiers (i), its class is determined as i. If the unknown data is not recognized by any classifier, it is considered as the hole. If the unknown data is recognized by more than two kinds of classifiers, handle it in the same way as the two classes, until all overlaps are eliminated.
2. Hole: Fig. 3a illustrates the hole state, in which the k-nearest neighbor algorithm is applied. We use $k = 3$ in our algorithm. The distance is defined as $dist(c, s) - c.r$,

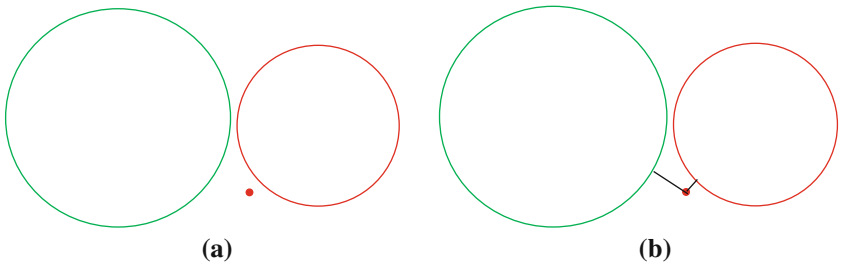

Fig. 3 **a** Hole. **b** The solution to hole

where $dist(c, s)$ is the Hamming distance between the classifier c and unknown data s. The data s is classified into the class where any two classifiers agree. If each of the three classifiers produces a different class, the data s is classified by the classifier with the nearest distance to it.

The classification is performed as follows:

Step 1. Select an unknown data and classify it using each classifier.
Step 2. If the unknown data is recognized by only one kind of classifiers (i), then it is classified as i, and go back to Step 1. Otherwise, if the state is overlap, go to Step 3. If the state is hole, go to Step 4.
Step 3. Classify the unknown data using solution to overlap.
Step 4. Classify the unknown data using solution to hole.

4 Evaluation

4.1 Compare the initial algorithm for classifier generation based on the maximum distance with the random initial classifier generation algorithm

In order to evaluate the performance of our algorithm, we compare the proposed algorithm to the random algorithm [29]. We use the sepallength and sepalwidth attributes of the Setosa and Versicolor classes in the Fisher's Iris data set [47] to formulate a two-dimension, two-class problem. The first 30 instances in each class are chosen for training data and the remaining 20 instances are used for test data. The parameters of PSCA are set to $\alpha = 0.85$, $N_g = 0$. The results are shown in Fig. 4. The red points represent Setosa, and the green points represent Versicolor. In Fig. 4, we can see that 2 classifiers for Setosa and 2 classifiers for Versicolor are generated by our algorithm. But 4 classifiers for Setosa and 3 classifiers for Versicolor are generated by the random algorithm. The two algorithms achieve the same accuracy of 97.5%.

We also compare these two algorithms using the entire Iris data set. Also, the first 30 instances of each class are chosen for training data and the remaining 20 instances are

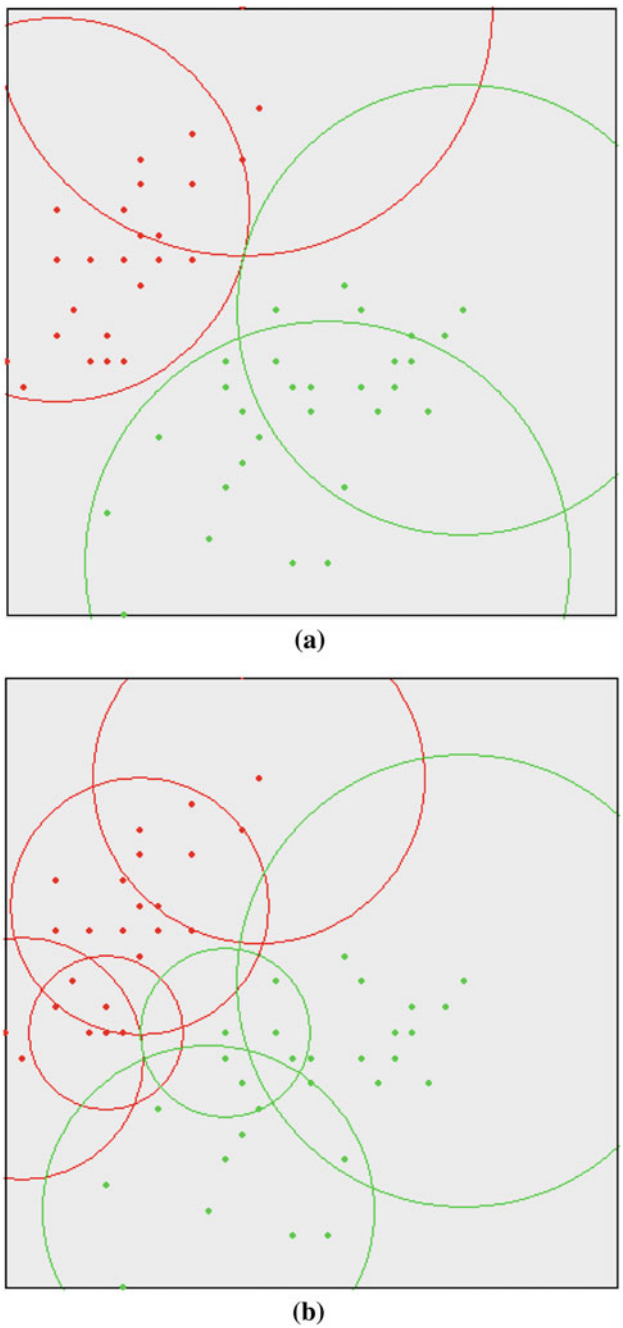

Fig. 4 **a** The generated classifiers by the initial algorithm for classifier generation based on the maximum distance. **b** The generated classifiers by the random initial classifier generation algorithm

used for test data. For our algorithm, the number of classifiers generated is 16, and the accuracy is 96.7%. For random algorithm, the number of classifiers generated is 17, and the accuracy is 95%. The results show that our algorithm generates fewer classifiers, but achieves higher accuracy than the random algorithm.

4.2 Dataset

We collect 3721 Windows malicious executables and 3458 benign Windows executables for experiment. 1421 malicious executables are randomly selected from VX Heavens [8], another 2300 are collected from Internet from December 2009 to April 2011. There are four types of malicious ones, including 912 backdoors (such as Backdoor.Win32.ColdDeath, Backdoor.Win32.Coma, Backdoor.Win32.Galaxy), 785 spyware (such as Spyware.Win32.RedCod.01, Spyware.Win32.AimSpy, Spyware.Win32.BC), 1,074 trojans (such as Trojan.Win32.FireKill, Trojan.Win32.Ghost, Trojan.Win32.PowerOff), and 950 worms (such as Worm.Win32.Burn.a, Worm.Win32.Denit, Worm.Win32.Elman). There are 343 variants in backdoors (such as Backdoor.Win32.Agent, Backdoor.Win32.Aimbot, Backdoor.Win32.BlackAngel, Backdoor.Win32.Ghost), 189 variants in spyware (such as Spyware.Win32.Agent, Spyware.Win32.Bancos, Spyware.Win32.Delf), 279 varians in trojans (such as Trojan.Win32.Agent, Trojan.Win32.PSW, Trojan.Win32.Downloader), and 365 variants in worms (such as Worm.Win32.Agent, Worm.Win32.AutoRun, Worm.Win32.Downloader). Table 1 shows the statistics of dataset.

Table 1 Statistics of dataset

File type	Quantity	Minimum size (KB)	Maximum size (KB)	Average size (KB)
Malware	3721	2	3,542	84
Benign	3458	9	263,525	138

For capturing IRPs, we developed MBMAS [20] based on kernel driver technology. The most advantage of MBMAS is that it can associate a process with its child processes. In run-time detection, if any child process is malicious then we believe that the father process and the executable which creates this process is malicious. Because some malware does not have malicious behavior in main process but creates child process, for example cmd.exe, to do. But MBMAS can not associate a thread with its children for now. We will fix it in the next version of MBMAS. Figure 5 is the user interface of MBMAS. Details of MBMAS can be found in [20].

The MBMAS is installed in a fresh VMware [37] virtual machine of Windows XP SP2 and a snapshot is taken. After each execution of malware or benign executable, we replace the virtual machine with the original snapshot.

In our previous work, we find 59 types of IRPs. We analyze IRP traces using 4-gram and shown in Fig. 6. We can see that there are less than 4,000 unique 4-grams in 600,000 IRPs.

For good performance, we use Information Gain (IG) [38] and Fisher score [39] to select top features for classification. Our previous work shows that 4-gram performs well. We test top 10, 50, 100, 200, 500, 1000 4-grams for IG and Fisher score and the top 500 4-grams outperform others. So we select the top 500 4-grams for IG and Fisher score to classify. We check each IRP sequence in testing dataset for presence or absence of the selected 4-grams. We place 1 if

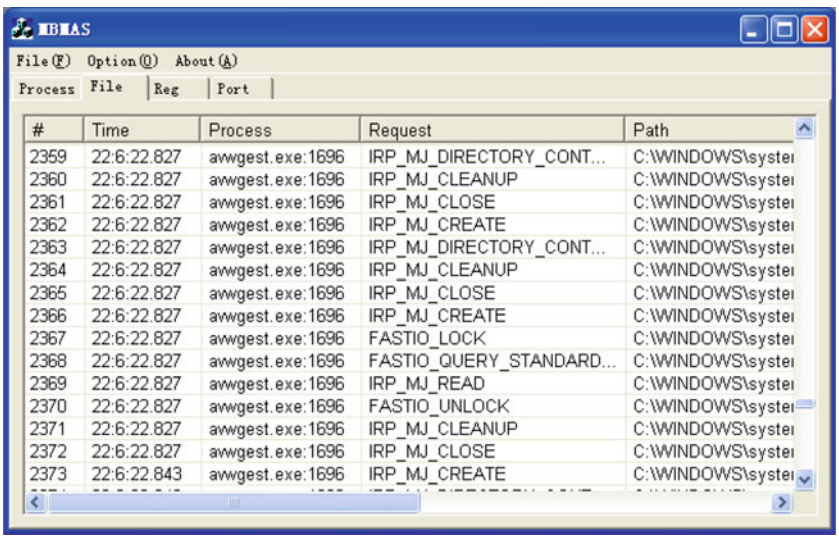

Fig. 5 The user interface of MBMAS

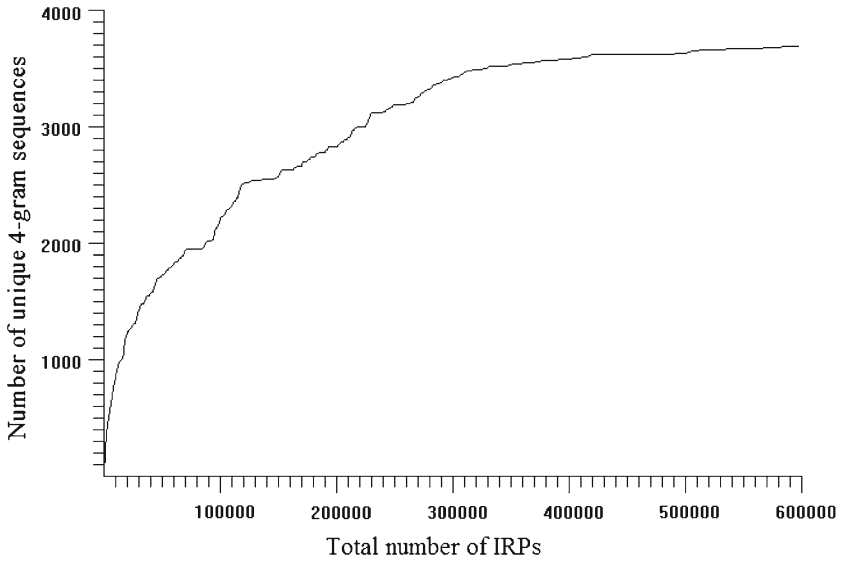

Fig. 6 Unique 4-gram sequences with the total number of IRPs growing

the 4-gram is present and 0 otherwise. Each IRP sequence is mapped to a 500-dimensional binary string for classification.

4.3 Evaluation measures

For evaluation purposes, we want to measure the accuracy of the classification algorithms, as well as the true positive rate and false positive rate. First, we give some definitions:

1. $|TP|$, the number of malicious executable samples classified as malicious executables.
2. $|TN|$, the number of benign programs classified as benign.
3. $|FP|$, the number of benign programs classified as malicious executables.
4. $|FN|$, the number of malicious executables classified as benign.

We use the common three measures to evaluate, which are the True Positive Rate (TRP), which is the rate of positive instances classified correctly, as shown in Eq. 3, False Positive Rate (FPR), which is the rate of negative instances misclassified, as shown in Eq. 4, and the Total Accuracy, which measures the number of absolutely correctly classified instances, either positive or negative, divided by the entire number of instances shown in Eq. 5.

$$TPR = \frac{|TP|}{|TP| + |FN|} \tag{3}$$

$$FPR = \frac{|FP|}{|FP| + |TN|} \tag{4}$$

$$Total\,Accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|} \tag{5}$$

We use 10-fold cross-validation to evaluate. That is, we randomly partitioned the candidate data set into ten disjoint sets of equal size, selected one as a testing set, and combined the remaining nine to form a training set. We conducted ten such runs using each partition as the testing set.

4.4 Sensitivity analysis of parameter α

Parameter α is the key of PSCA. That's important to know the impact of α on accuracy. Set $N_g = N_c = 100$, $P = 0.2$, and all results are obtained by averaging ten runs of each value of α. Figure 7 shows the impact of α on accuracy with different feature selection algorithms. For IG, the accuracy is not sensitive to the change of α. The minimum value is 97.38% when $\alpha = 0.2$, and the maximum value is 99.22% when $\alpha = 0.55$ or 0.65. But for Fisher score, the accuracy is more than 97% when α from 0.3 to 0.75, and less than 72% when α less than 0.3 or more than 0.75. The maximum value is 99.30% when $\alpha = 0.6$ or 0.65. So parameter α should be adjusted for different features to obtain an optimal result.

4.5 Sensitivity analysis of parameter N_g and N_c

Set $\alpha = 0.65$, $N_g = N_c$, and $P = 0.2$. The results are obtained by averaging ten runs of each value of parameters N_g and N_c. Figure 8 shows the impact of N_g and N_c on accuracy. Results reveal that the clonal selection does not always produce an increase in accuracy. So the initial algorithm for classifier generation based on the maximum distance can achieve approximate optimal results. The clonal selection is used to search optimal results, but can not be used to retrain the classifiers. The mutation probability P also has a little help to improve performance, but can not guarantee the improvement of performance. If the value of

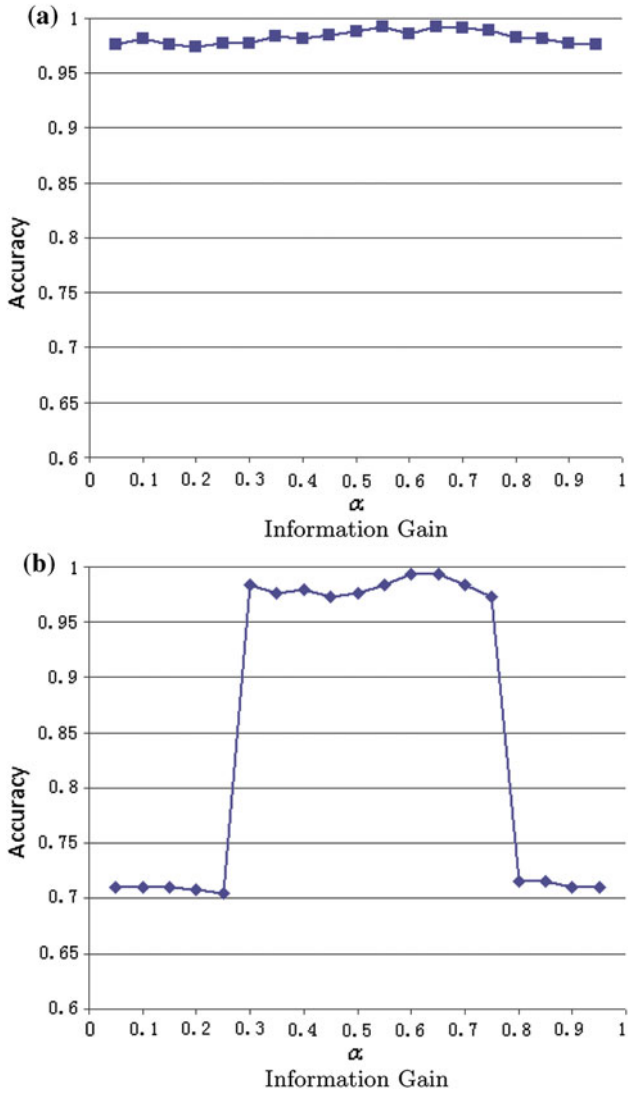

Fig. 7 The impact of α on accuracy

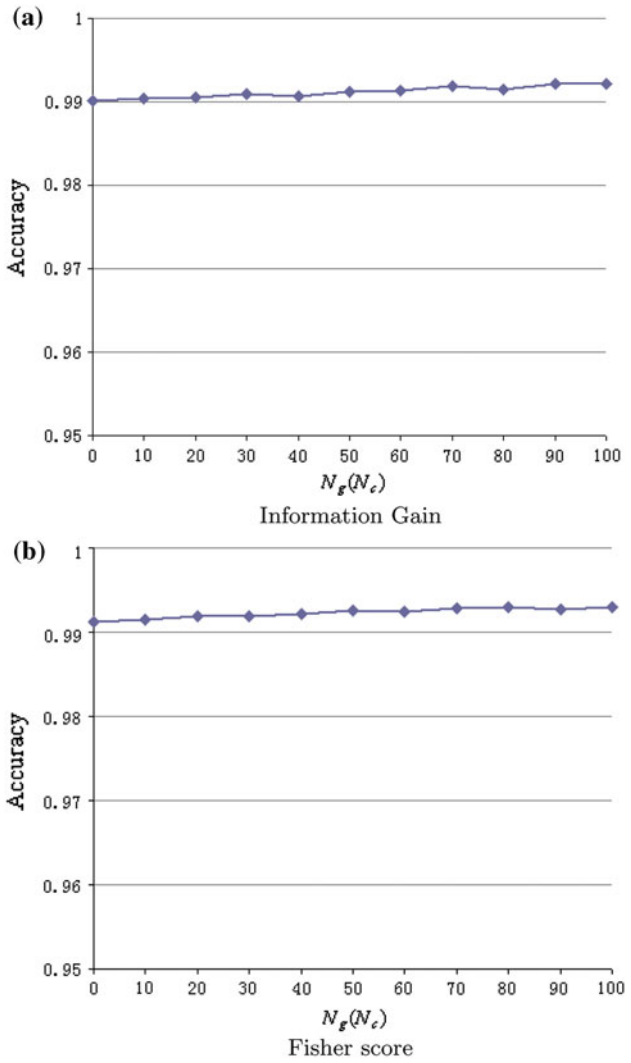

Fig. 8 The impact of N_g and N_c on accuracy

P increase, more learning time is needed. So the best value of P is 0.2 or 0.3.

4.6 Comparison

In this section, we compare results of PSCA with ANSC, NB, BN, SVM, DT, BNB, BBN, BSVM, and BDT. For NB, we use NaiveBayes implemented in WEKA [40,41]. For BN, we use WEKA's BayesNet. For SVM, we use sequential minimal optimization (SMO) [42] implemented in WEKA. For DT, we use J48 implementation in WEKA. And for Boosting, we use the AdaBoost.M1 algorithm [43] implemented in WEKA. We use CVParameterSelection to select the optimal parameters of algorithms implemented in WEKA. In ANSC, set $\alpha = 0.65$, $P_m = 0.2$, $N_g = N_c = 100$, and the cutting method is not used. In PSCA, set $\alpha = 0.65$, $P = 0.2$, and $N_g = N_c = 100$. Results are shown in Table 2 (Remark IG-NB Naïve Bayes with Information Gain, FS-NB Naïve Bayes with Fisher score, and so on), and the highest TPR, highest Total Accuracy and lowest FPR are bolded.

Among these methods, FS-PSCA outperforms others with the highest accuracy of 99.30%. The TPR of FS-PSCA is 98.87% the same as IG-PSCA, heigher than ANSC and other methods. The FPR of FS-PSCA is 0.23%. In general, PSCA outperforms other well known methods. We also see that the performances of classification algorithms are influenced by different feature selection algorithms. Good feature selection can help classification.

We also present the average number of classifiers in PSCA and ANSC. In PSCA, the average number of classifiers is 134.4 for malware and 64.3 for benign. In ANSC, the average number of classifiers is 168.2 for malware and 102.4 for benign. Fewer classifiers are needed in PSCA than ANSC. This reduction is caused by the initial algorithm for classifier generation based on the maximum distance. Because this algorithm can cover more self-space with fewer classifiers

Table 2 Results of all algorithms

	TPR (%)	FPR (%)	Total accuracy (%)
IG-NB	83.53	1.68	90.65
IG-BN	79.06	2.00	86.38
IG-SVM	97.69	2.02	97.83
IG-DT	94.68	4.25	95.19
IG-BNB	97.85	3.90	97.01
IG-BBN	94.09	2.54	95.71
IG-BSVM	97.69	2.02	97.83
IG-BDT	96.41	3.59	96.42
IG-ANSC	98.58	0.49	99.02
IG-PSCA	**98.87**	0.40	99.22
FS-NB	96.18	**0.20**	97.92
FS-BN	91.16	0.23	95.31
FS-SVM	96.96	1.76	97.58
FS-DT	96.45	1.65	97.37
FS-BNB	97.04	0.46	98.24
FS-BBN	97.53	0.26	98.59
FS-BSVM	96.96	1.76	97.57
FS-BDT	98.82	0.38	99.21
FS-ANSC	98.63	0.38	99.11
FS-PSCA	**98.87**	0.23	**99.30**

than ANSC. The detection time is proportional to the number of classifiers. So PSCA is more effective than ANSC.

4.7 Discussion

There is a problem that the IRP traces of programs are vary from a host to another. But the functions of programs are the same. After extensive analysis we found that the difference between two IRP traces of the same program is that sometimes an IRP is not successful, so the IRP will keep trying until success. So the real difference between the two IRP traces is that some IRPs repeat some times. This difference has little effect on the results. The system does not need to be retrained every time new programs are installed.

5 Conclusion

In this paper, we propose a novel classification algorithm PSCA for malware detection based on positive selection. For general purpose, PSCA not just work with two-class classification problems, but multi-class classification problems. PSCA turns multi-class classification problem into two-class classification problem: self and nonself. Only the self-class classifiers are generated for classification. The initial algorithm for classifier generation based on the maximum distance, which is able to produce approximate optimal initial classifiers, is proposed. The clonal selection algorithm is used in PSCA to search optimal classifiers. K-nearest neighbor algorithm is used to solve the hole problem to avoid more than one enlarged detectors covers the self-space. For the problem that the API call sequences can be manipulated to circumvent detection, we use more effective and robust technology, kernel mode callbacks, to monitor program behavior. In our malware detection experiments, PSCA outperforms ANSC, Naïve Bayes, Bayesian Networks, Support Vector Machine, and Decision Tree, which are excellent general classification algorithms.

Acknowledgments The work presented in this paper is supported by the National Natural Science Foundation of China under Grant No. 61070015, the National Technical Innovation Foundation of China under Grant No. 08C26214411198, and the Guangdong-Hong Kong Technology Cooperation Funding Scheme of China under Grant No. 2008A011400010.

References

1. Symantec Corporation.: Internet security threat report volume XV. http://www.symantec.com/business/theme.jsp?themeid=threatreport
2. Willems, C., Holzand, T., Freiling, F.: Toward automated dynamic malware analysis using CWSandbox. IEEE Secur. Priv. **5**(2), 32–39 (2007)
3. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A sense of self for Unix processes. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 120–128 (1996)
4. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. J. Comput. Secur. **6**(3), 151–180 (1998)
5. Wespi, A., Dacier, M., Debar, H.: Intrusion detection using variable-length audit trail patterns. In: Proceedings of the Recent Advances in Intrusion Detection, pp. 110–129. Springer, France (2000)
6. Sato, I., Okazaki, Y., Goto, S.: An improved intrusion detection method based on process profiling. IPSJ J. **43**, 3316–3326 (2002)
7. Manzoor, S., Shafiq, M.Z., Tabish, S.M., Farooq, M.: A sense of 'danger' for windows processes. In: ICARIS. LNCS, vol. 5666, pp. 220–233. Springer, Heidelberg (2009)
8. VX Heavens Virus Collection. http://vx.netlux.org/vl.php
9. API Monitor. http://www.rohitab.com/apimonitor
10. Aickelin, U., Bentley, P., Cayzer, S., Kim, J., McLeod, J.: Danger theory: the link between AIS and IDS? In: Proceedings of the ICARIS. LNCS, vol. 2787, pp. 147–155, Springer, Heidelberg (2003)
11. Greensmith, J., Aickelin, U., Cayzer, S.: Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In: Proceedings of the ICARIS. LNCS, vol. 3627, pp. 153–167, Springer, Heidelberg (2005)
12. Greensmith, J., Aickelin, U.: The deterministic dendritic cell algorithm. In: Proceedings of the ICARIS. LNCS, vol. 5132, pp. 291–303. Springer, Heidelberg (2008)
13. Ahmed, F., Hameed, H., Shafiq, M.Z., Farooq, M.: Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 55–62 (2009)

14. Parampalli, C., Sekar, R., Johnson, R.: A practical mimicry attack against powerful system-call monitors. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security (AsiaCCS), pp. 156–167, Japan (2008)
15. Wagner, D., Soto, P.: Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS), pp. 255–264. ACM Press, New York (2002)
16. Oberheide, J.: Detecting and evading CWSandbox. http://jon.oberheide.org/blog/2008/01/15/detecting-and-evading-cwsandbox/
17. Seifert, C., Steenson, R., Welch, I., Komisarczuk, P., Endicott-Popovsky, B.: Capture—a behavioral analysis tool for applications and documents. Digit. Investig. **4**(Suppl. 1), S23–S30 (2007)
18. Bassov, A.: Hooking the kernel directly. http://www.codeproject.com/system/soviet_direct_hooking.asp
19. Field, S.: An introduction to kernel patch protection. http://blogs.msdn.com/windowsvistasecurity/archive/2006/08/11/695993.aspx
20. Zhang, F.Y., Qi, D.Y., Hu, J.L.: MBMAS: a system for malware behavior monitor and analysis. In: Proceedings of the International Symposium on Computer Network and Multimedia Technology, pp. 1–4 (2009)
21. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 202–212 (1994)
22. Forrest, S., Hofmeyr, S.A., Somayaji, A.: Computer immunology. Commun. ACM. **40**(10), 88–96 (1997)
23. Esponda, F., Forrest, S., Helman, P.: A formal framework for positive and negative detection schemes. IEEE Trans. Syst. Man Cybern. B **34**(1), 357–373 (2004)
24. de Castro, L.N., Von Zuden, F.J.: Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comput. **6**(3), 239–251 (2002)
25. Coello, C.A.C., Rivera, D.C., Cortes, N.C.: Use of an artificial immune system for job shop scheduling. LNCS, vol. 2787, pp. 1–10 (2003)
26. de Castro, L.N., Von Zuden, F.J.: aiNet: an artificial immune network for data analysis. In: Data Mining: A Heuristic Approach. Idea Group Publishing, USA (2001)
27. Neal, M.: Meta-stable memory in an artificial immune network. In: Proceedings of ICARIS 2003, pp. 168–181 (2003)
28. Watkins, A., Timmis, J., Boggess, L.: Artificial immune recognition system (AIRS): an immune-inspired supervised learning algorithm. Genet. Program. Evol. Mach. **5**(3), 291–317 (2004)
29. Igawa, K., Ohashi, H.: A negative selection algorithm for classification and reduction of the noise effect. Appl. Soft Comput. **9**(1), 431–438 (2009)
30. Kahramanli, H., Allahverdi, N.: Extracting rules for classification problems: AIS based approach. Expert Syst. Appl. **36**(7), 10494–10502 (2009)
31. de Castro, L.N., Von Zuben, F.J.: The clonal selection algorithm with engineering applications. In: Proceedings of the 2000 GECCO, Workshop on Artificial Immune Systems and Their Applications, pp. 36–37. Morgan Kaufmann, San Francisco (2000)
32. Seiden, P.E., Celada, F.: A model for simulating cognate recognition and response in the immune system. J. Theor. Biol. **158**(3), 329–357 (1992)
33. Sim, K.-B., Lee, D.-W.: Modeling of positive selection for the development of a computer immune system and a self-recognition algorithm. Int. J. Control Autom. Syst. **1**(4), 453–458 (2003)
34. Dervovic, D., Zuniga-Pflucker, J.C.: Positive selection of T cells, an in vitro view. Semin. Immunol. **22**(5), 276–286 (2010)
35. Yang, S.Y., Wang, M., Jiao, L.C.: Quantum-inspired immune clone algorithm and multiscale Bandelet based image representation. Pattern Recognit. Lett. **31**(13), 1894–1902 (2010)
36. Laurentys, C.A., Ronacher, G., Palhares, R.M., Caminhas, W.M.: Design of an artificial immune system for fault detection: a negative selection approach. Exp. Syst. Appl. **37**(7), 5507–5513 (2010)
37. VMware. http://www.VMware.com
38. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. **7**, 2721–2744 (2006)
39. Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A., Bloomfield, C.D., Lander, E.S.: Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science **286**(5439), 531–537 (1999)
40. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Elsevier, San Francisco (2006)
41. Weka. http://www.cs.waikato.ac.nz/ml/weka/
42. Platt, J.: Fast training of support vector machines using sequential minimal optimization. In: Schölkopf, B., Burges, C., Mika, S. Advances in Kernel Methods—Support Vector Learning, MIT Press, Cambridge (1998)
43. Freund, Y., Schapire, R.: Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on Machine Learning, pp. 148–156 (1996)
44. Aydin, I., Karakose, M., Akin, E.: Chaotic-based hybrid negative selection algorithm and its applications in fault and anomaly detection. Exp. Syst. Appl. **37**(7), 5285–5294 (2010)
45. Gao, X.Z., Ovaska, S.J., Wang, X.: Particle swarm optimization of detectors in negative selection algorithm. In: Proceedings of IEEE Systems Man Cybernetics, Montreal, Quebec, Canada, pp. 1236–1242 (2007)
46. Zhou, J., Dipankar, D.: Real-valued negative selection algorithm with variable sized detectors. In: Proceedings of Genetic and Evolutionary Computation Conference, vol. 3102, pp. 287–298 (2004)
47. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/