VNU Journal of Science: Comp. Science & Com. Eng. Vol. 31, No. 1 (2015) 22–31

A Novel Combination of Negative and Positive Selection in Artificial Immune Systems

Van Truong Nguyen[1], Xuan Hoai Nguyen[2], Chi Mai Luong[3]

[1]*Thai Nguyen University of Education, Thai Nguyen, Vietnam*
[2]*Hanoi University, Hanoi, Vietnam*
[3]*Vietnamese Academy of Science and Technology, Hanoi, Vietnam*

Abstract

Artificial Immune System (AIS) is a multidisciplinary research area that combines the principles of immunology and computation. Negative Selection Algorithms (NSA) is one of the most popular models of AIS mainly designed for one-class learning problems such as anomaly detection. Positive Selection Algorithms (PSA) is the twin brother of NSA with quite similar performance for AIS. Both NSAs and PSAs comprise of two phases: generating a set D of detectors from a given set S of selves (detector generation phase); and then detecting if a given cell (new data instance) is self or non-self using the generated detector set (detection phase). In this paper, we propose a novel approach to combining NSAs and PSAs that employ binary representation and r-chunk matching rule. The new algorithm achieves smaller detector storage complexity and potentially better detection time in comparison with single NSAs or PSAs.

Manuscript communication: received 17 February 2014, revised 01 March 2015, accepted 25 March 2015
Corresponding author: Van Truong Nguyen, nvtruongtn@gmail.com

Keywords: Artificial immune systems, Negative selection, Positive selection, Intrusion detection, Detector

1. Introduction

The biological immune system is a mature defense system which has evolved over millions of years. As a defense system, it is incredibly robust, adaptive, and inherently distributed. The immune system posses powerful pattern recognition, learning, and memory capabilities. It has evolved complex methods for combating infections caused by viruses and other pathogens, without apparently any central coordination or control. Its ability to distinguish between pathogens and non-pathogens has inspired a number of artificial immune systems on computers [1]. The representative immune cell is the T cell, which has a self-recognition component and an antigen receptor for locating and eliminating infected pathogens. The learning process of the biological immune system does not require negative examples and acquired knowledge is represented in an explicit form: T cells are generated randomly and in a large number, in the hope that every pathogen that might infect the host could be detected by at least some of these cells. However, the host must ensure that no cell generated would turn against itself (autoimmune reactions). Hence, newborn T cells undergo the process of selection to ensure that they are able to recognize non-self peptides. This process might be conducted in two ways: positive selection and negative selection. In negative selection, if a T cell detects any self protein, it is discarded; otherwise, it is kept. By contrast, in positive selection, if a T cell fails to recognize any of the self proteins, it is removed [2].

Negative selection algorithms (NSA) and positive selection algorithm (PSA) are computational models that have been inspired by negative and positive selection of the biological immune system. Among the two, NSA has been studied more extensively resulting in more variants and applications [3]. However, all of existing NSAs have worst-case exponential memory complexity for storing the detector set, hence, limit their practical applicabilities [4]. In this paper, we propose a novel selection algorithm that employs binary representation and *r*-chunk matching rule for detectors. The new algorithm combines negative and positive selection to reduce both detector storage complexity and detection time, while maintaining the same detection coverage as that of NSAs (PSAs).

The rest of the paper is organized as follows. In the next section, we present the background on PSAs, NSAs and *r*-chunk matching rule for detectors. Section 3 briefly describes the work in the literature that are most related to our new approach. Section 4 details our new selection algorithm with theoretically proven results on detector storage optimization and preliminary experimental results on detection time. Section 5 concludes the paper and discuss some possible future work.

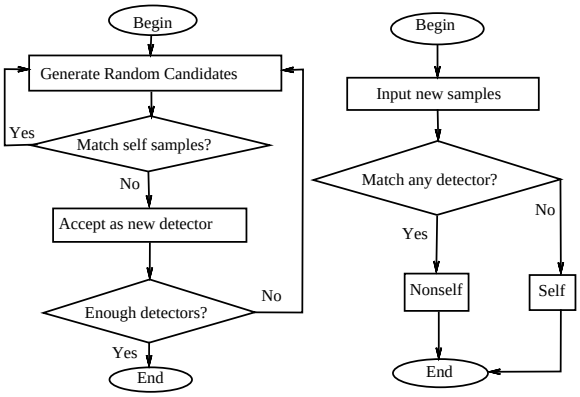
The main contributions of this paper, compared to our previous work [5], are three-folds: a more general proof of detector storage complexity, an extension of related works, and an experiment of our algorithm on real network intrusion dataset.

2. Background

In this section we first briefly describe NSAs and PSAs. Then, the *r*-chunk matching rule is defined and discussed.

2.1. Negative Selection Algorithms

NSAs are among the most popular and extensively studied techniques in artificial immune systems that simulate the negative selection process of the biological immune system. A typical NSA comprises of two phases: detector generation and detection [6, 7]. In the detector generation phase (Figure. 1.a), the detector candidates are generated by some random processes and censored by matching them against given self samples taken from a set S (representing the system components). The candidates that match any element of S are eliminated and the rest are kept and stored in the detector set D. In the detection phase (Figure. 1.b), the collection of detectors are used to distinguish self (system components) from non-self (outlier, anomaly, etc). If incoming data instance matches any detector, it is claimed as non-self or anomaly.



(a) Generation of detector set (b) Detection of new instances

Fig. 1: Outline of a typical negative selection algorithm.

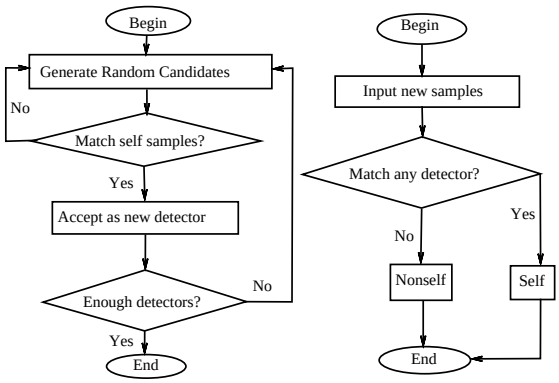
Since its introduction, NSA has had many applications such as in computer virus detection [8][9], monitoring UNIX processes [10], anomaly detection in time series [11], intrusion detection [2], scheduling [12], fault detection and diagnosis [13].

2.2. Positive Selection Algorithms

Contrary to NSAs, PSAs have been less studied in the literature. They are mainly developed and applied in intrusion detection [14], spam detection [15], and classification [16]. Stibor et al. [1] argued that positive selection might have better detection performance than negative selection. However, for problems and applications that the number of detectors generated by negative selection algorithms is

much less than the number of self samples, negative selection is obviously a better choice [3].

Similar to NSA, a PSA contains two phases: detector generation and detection. In the detector generation phase (Figure. 2.a), the detector candidates are generated by some random processes and matched against the given self sample set S. The candidates that do not match any element in S are eliminated and the rest are kept and stored in the detector set D. In the detection phase (Fig. 2.b), the collection of detectors are used to distinguish self from non-self. If incoming data instance matches any detector, it is claimed as self.



(a) Generation of detector set (b) Detection of new instances

Fig. 2: Outline of a typical positive selection algorithm.

2.3. Positive and Negative r-chunk Detectors

In PSAs and NSAs, an essential component is the matching rule which determines the similarity between detectors and self samples (in the detector generation phase) and coming data instances (in the detection phase). Obviously, the matching rule is dependent on detector representation. In this paper, we assume binary representation for all detectors and data. Binary representation is the most simple and popular representation for detectors and data in AIS, and other representations (such as real valued) could be reduced to binary [17, 3]. For binary based AIS, the r-chunk and r-contiguous detectors are among the most common matching rules. An r-chunk matching rule can be seen as a generalisation of the r-contiguous matching rule, which helps AIS to achieve better results on data where adjacent regions of the input data sequence are not necessarily semantically correlated, such as in network data packets [18].

We denote $\Sigma = \{0, 1\}$ as the alphabet for detectors and data. Let $s \in \Sigma^\ell$ be a binary string, $\ell = |s|$ is the length of s and $s[i, \ldots, j]$ is the substring of s that starts at position i with length $j - i + 1$. Positive and negative r-chunk detectors could be defined as follows:

Definition 1 (Positive r-chunk detectors)**.** *Given a self set S, a tuple (d, i) of a string $d \in \Sigma^r$, $r \leq \ell$, and an integer $i \in \{1, ..., \ell - r + 1\}$ is called a positive r-chunk detector if there exists a $s \in S$ such that d matches $s[i, \ldots, i + r - 1]$.*

Definition 2 (Negative r-chunk detectors)**.** *Given a self set S, a tuple (d, i) of a string $d \in \Sigma^r$, $r \leq \ell$, and an integer $i \in \{1, ..., \ell - r + 1\}$ is called a negative r-chunk detector if d does not match any $s[i, \ldots, i + r - 1]$, $s \in S$.*

We also use the following notations:

- $Dp_i = \{(d, i), (d, i)$ is a positive r-chunk detector$\}$ is set of all positive r-chunk detectors at position i, $i = 1, \ldots, \ell - r + 1$.
- $Dn_i = \{(d, i), (d, i)$ is a negative r-chunk detector$\}$ is set of all negative r-chunk detectors at position i, $i = 1, \ldots, \ell - r + 1$.
- $Dp = \cup_{i=1}^{\ell-r+1} Dp_i$ is set of all positive r-chunk detectors.
- $Dn = \cup_{i=1}^{\ell-r+1} Dn_i$ is set of all negative r-chunk detectors.
- For a given detector set X, S_X and N_X are the sets of self and non-self strings detected by X, respectively.

Example 1. *Let $\ell = 5$, matching threshold $r = 3$. Suppose that we have the set of six self strings $S = \{s_1 = 00000; s_2 = 00010; s_3 = 10110; s_4 = 10111; s_5 = 11000; s_6 = 11010\}$. $Dp_1 = \{(000,1); (101,1); (110,1)\}$ (Dp_1 is set of all leftmost substring of length ℓ of s, $s \in S$), $Dn_1 = \{(001,1); (010,1); (011,1); (100,1); (111,1)\}$,*

Dp_2 = {(000,2); (001,2); (011,2); (100,2); (101,2)}, Dn_2 = {(010,2); (110,2); (111,2)}, Dp_3 = {(000,3); (010,3); (110,3); (111,3)}, Dn_3 = {(001,3); (011,3); (100,3); (101,3)} (note that $Dp_i \cup Dn_i = \Sigma^3$, $i = 1, 2, 3$). The self space covered by the set of all positive 3-chunk detectors is S_{Dp} = {00000; 00001; 00010; 00011; 00110; 00111; 01000; 01001; 01010; 01011; 01110; 01111; 10000; 10001; 10010; 10011; 10100; 10101; 10110; 10111; 11000; 11001; 11010; 11011; 11110; 11111}. The non-self strings detected by the set of all negative 3-chunk detectors is N_{Dn} = {00001; 00011; 00100; 00101; 00110; 00111; 01000; 01001; 01010; 01011; 01100; 01101; 01110; 01111; 10000; 10001; 10010; 10011; 10100; 10101; 11001; 11011; 11100; 11101; 11110; 11111}.

It could be seen from Example 1 that $N_{Dp} = \Sigma^5 \backslash S_{Dp}$ = {00100; 00101; 01100; 01101; 11100; 11101} $\neq$ N_{Dn}, so the detection coverage of *Dn* is not the same as that of *Dp*. This is undesirable for the combination of PSA and NSA. Hence, to combine positive and negative selection algorithms in an unified framework, we have to change the semantics of positive selection in the detection phase as follows.

Definition 3 (Detection in positive selection)**.** *If new instance matches $\ell - r + 1$ positive r-chunk detectors (d_{ij}, i), $i = 1, \ldots, \ell - r + 1$, it is claimed as self, otherwise it is claimed as non-self.*

With the new detection semantics, the following proposition on the equivalence of detection coverage of *r*-chunk type PSA and NSA could be stated.

Proposition 1 (Detection Coverage)**.** *The detection coverage of positive and negative selection algorithms coincide.*

$$N_{Dp} = N_{Dn} \tag{1}$$

$$S_{Dp} = S_{Dn} \tag{2}$$

Proof. From the description of NSAs (see Fig. 1), if a new data instance matches a negative *r*-chunk detector, then it is claimed as non-self, otherwise it is claimed as self. Obviously, it is dual to the detection of new data instances in positive selection as given in Definition 3. □

This proposition lays the foundation for our novel Positive-Negative Selection Algorithm (PNSA) proposed in Section 4.

3. Related Works

Both PSA and NSA achieve quite similar performance for detecting novelty in data patterns [19]. Dasgupta D. et al. [20] conducted one of the earliest experiments on combining positive with negative selection. The combined process is embedded in a genetic algorithm using a fitness function that assigns a weight to each bit based on the domain knowledge. Their method is neither aimed to reduce detector storage complexity nor detection time. Esponda et al. [21] proposed a generic NSA for anomaly detection problems. Their model of normal behavior is constructed from an observed sample of normally occurring patterns. Such a model could represent either the set of allowed patterns (positive detection) or the set of anomalous patterns (negative detection/selection). However, their NSA is not concerned with the combination of positive and negative selection in detection phase as in PNSA. Stibor et al. [1] argued that positive selection might have better detection performance than negative selection. However, the choice between positive selection algorithms and negative ones obviously depends on representation of the AIS-based applications. An example in Section 4 shows that some positive trees are more compact than the others and vice versa.

To the best of our knowledge, there has not been any published attempt in combining *r*-chunk type PSA and NSA for the purpose of reducing detector storage complexity and real/average detection time complexity.

4. New Positive-Negative Selection Algorithm

It can be seen from Section 2 that the positive and negative selection are dual. This motivates

our approach to combining the two mechanisms. In this section, a new r-chunk type NSA is proposed that is the combination of positive and negative selection.

In our proposed approach, binary trees are used as data structure for storing the detector set to reduce memory complexity, and therefore the time complexity of the detection phase. To build and store the detection set, our algorithm first constructs $\ell - r + 1$ binary trees (called positive trees) corresponding to $\ell - r + 1$ positive r-chunk detector set Dp_i, $i = 1, \ldots, \ell - r + 1$. Then, all complete subtrees of these trees are removed to achieve a compact representation of the positive r-chunk detector set while maintaining the detection coverage. Finally, for every i^{th} positive trees, we decide whether or not it should be converted to the negative tree, which covers the negative r-chunk detector set Dn_i. The decision depends on which tree is more compact. When this process is done, we have $\ell - r + 1$ compact binary trees that some of them represent positive r-chunk detectors and the others represent negative ones.

The r-chunk matching rule on binary trees is implemented as follows: a given sample s matches the positive (negative) tree i^{th} if $s[i, \ldots, i + k]$ is a path from the root to a leaf, $i = 1, \ldots, \ell - r + 1$, $k < r$. The detection phase can be conducted by traveling the compact binary trees iteratively one by one: a sample s is claimed as non-self if it matches a negative tree or it does not match all positive trees, otherwise it is considered as self.

Example 2. *For the set of self strings S from Example 1, where $\ell = 5$ and $r = 3$, the six binary trees (the left and right child are labeled 0 and 1 respectively) represent the detector set of six 3-chunk detectors (Dp_i and Dn_i, $i = 1, 2, 3$) as depicted in Figure 3. In the Figure, dashed arrows in some positive trees mark the complete subtrees that will be removed to achieve compact tree representation.*

The number of nodes of the trees in Figures 3.a - 3.f (after deleting complete subtrees) are 9, 10, 7, 6, 8 and 8, respectively. Therefore, the chosen final trees are those in Figures 3.a (9 nodes), 3.d (6 nodes) and 3.e or 3.f (8 nodes). In real implementation, it is unnecessary to generate both positive and negative trees. Since each Dp_i could dually be represented either by a positive or a negative tree, we only need to generate (compact) positive trees. If a compact positive tree T has more number of leaves than the number of internal nodes that have single child, the corresponding negative tree NT should have less number of nodes than T. Therefore, NT should be used instead of T to represent Dn_i more compactly. It is noted that NT could be obtained from T by taking the dual links (paths) in T. The following example illustrates this observation.

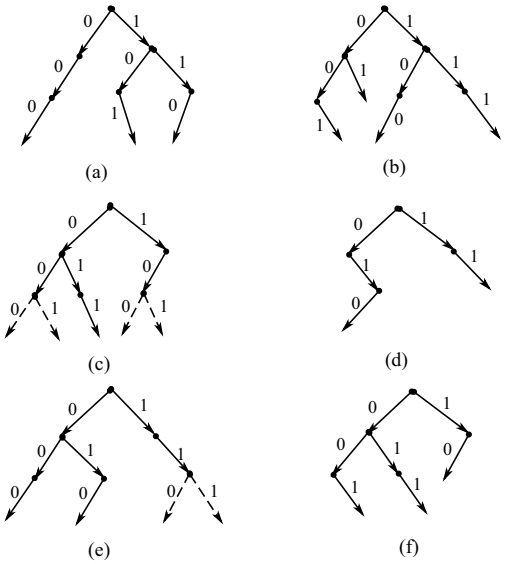


Fig. 3: Binary tree representation of the detector set generated from S defined in Example 1. The positive trees for Dp_1, Dp_2 and Dp_3 are in (a), (c) and (e), respectively; The negative trees for Dn_1, Dn_2 and Dn_3 are in (b), (d) and (f), respectively.

Example 3. *Consider again the set of self strings S from Example 1. The compact positive tree for the positive 3-chunk detector set Dp_2 = {(000,2); (001,2); (011,2); (100,2); (101,2)} is shown in Fig. 4.a. This tree has three leaves and two nodes that have only one child (in dotted circles) so it should be converted to the corresponding negative tree as illustrated in Fig. 4.b.*

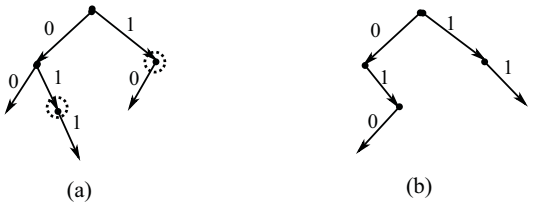


Fig. 4: Conversion of a positive tree to a negative one.

Algorithm 38 summarizes the overall PNSA. In the algorithm, the process of generating compact binary (positive and negative) trees representing the complete r-chunk detector set is conducted in the outer "for" loop. First, all binary positive tree T_i are constructed by the first inner loop. Then, the compactification of all T_i are conducted by the second one, $i = 1, \ldots, \ell - r + 1$. The conversion of a positive tree to negative one takes place in "if" statement after the second inner "for" loop. The procedure for recognizing a given cell string s^* as self or non-self, is carried out by the last "while ... do" and "if ... then ... else" statements.

The detection phase of PNSA could be illustrated by the following example.

Example 4. *Given S, r as in Example 1, and $s^* = 10100$ as the inputs of the algorithm, three binary trees are constructed as the detector set in Figures 3.a, 3.d. and 3.e. The output of the algorithm is "s^* is non-self" because all the paths of tree T_2 do not contain substring $s^*[2,\ldots,4] = 010$ of s^*.*

From the description of PNSA, it is trivial to show that it takes $|S|(\ell - r + 1).r$ steps to generate all necessary trees (detector generation time complexity) and $(\ell - r + 1).r$ steps to verify a cell string as self or non-self in the worst case (worse-case detection time complexity). These time complexities are similar to the state-of-the-art NSAs (PSAs) such as the one proposed in [4]. However, by using compact positive and negative binary trees for storing the detector set, PNSA could reduce the storage complexity of the detector set in comparison with the other r-chunk type single NSAs or PSAs that store detectors as binary strings. This storage complexity reduction could potentially lead to better detection time complexity in real and average cases. To see this, first, let the following theorem be stated:

Theorem 1 (PNSA detector storage complexity)**.** *Given a self set S and an integer ℓ, the PNSA produces the detector (binary) tree set that have at most total $(\ell - r + 1).2^{r-2}$ less number of nodes in comparison to the detector tree set created by a PSA or NSA only, where $r \in \{2, \ldots, \ell - r + 1\}$.*

Proof. We only prove the theorem for the PSA case, the NSA case can be proven in a similar way. Because there are $(\ell - r + 1)$ positive trees can be build from the self set S, so the theorem is proved if it can reduce at most 2^{r-2} nodes from a positive tree.

The theorem is proved by induction on r (also the height of binary trees).

It is noted that when converting a positive tree to a negative tree as in PNSA, the reduction in number of nodes is exactly as the result of the subtraction of number of leaf nodes from the number of internal nodes that have only one child.

When $r = 2$, there are 16 trees of possible positive trees are of height 2. By examining all 16 cases, we have found that the maximum reduction in number of nodes is 1. One example of these cases is the positive tree that has 2 leaf nodes after compactification as in Fig. 5.a. Since it has two leaf nodes and one one-child internal node, after being converted to the corresponding negative tree, the number of nodes is reduced by 2 - 1 = 1.

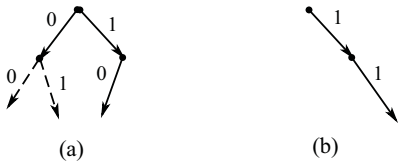


Fig. 5: One node is reduced in a tree: a compact positive tree has 4 nodes (a) and its conversion (a negative tree) has 3 node (b).

Suppose that the theorem's conclusion is right for all $r < k$. We shall prove that it is also right for k. This is done by an observation that in all positive trees that are of height k, there is at least

Algorithm 1: PNSA (Positive-Negative Selection Algorithm)

Data: a self set S, an integer $r \in \{1, \ldots, \ell - r + 1\}$
a cell string s^* to be detected.
Result: detection of s^* as self or non-self.

for $i = 1, \ldots, \ell - r + 1$ **do**
  initialize an empty binary self tree T_i
  for *each* $s \in S$ **do**
    insert $s[i, \ldots, r - i + 1]$ into T_i
  end
  for *every internal node* $n \in T_i$ **do**
    if *n is root of complete binary subtree* **then**
      delete this subtree
    end
  end
  if *(number of leaves of* T_i*) > (number of nodes of* T_i *that have only one child)* **then**
    for *every internal node* $\in T_i$ **do**
      if *it has only one child* **then**
        if *the child is a leaf* **then**
          delete the child
        end
        create the other child for it
      end
    end
    Mark T_i as a negative tree
  end
end
$flag$ = true
$i = 1$
while $(i \leq \ell - r + 1)$ *and* $(flag =$ *true)* **do**
  if *(*T_i *is positive tree) and (*s^* *does not match* T_i*)* **then**
    $flag$ = false;
  end
  if *(*T_i *is negative tree) and (*s^* *matches* T_i*)* **then**
    $flag$ = false
  end
  i=i+1
end
if $flag =$ *false* **then**
  output "s^* is non-self"
else
  output "s^* is self"
end

one tree with both left subtree and right subtree (of height $k - 1$) that each can be reduced by at least $2^{(k-1)-2}$ nodes after conversion. □

A real experiment on network intrusion dataset at the end of this section shows that the storage reduction is only about 0.35% of this maximum.

Next, we investigate the possible impact of the reduction in detector storage complexity in PNSA on the detection real (average) time complexity in comparison with single NSA (PSA). Figure 6 shows the results on detector memory storage and detection time of PNSA compared to one of the state-of-the-art single NSAs proposed in [4] on some combinations of S, ℓ and r. The training data set of selves S contains randomly generated binary strings. The memory reduction is measured as the ratio of reduction in number of nodes of the binary tree detectors generated by PNSA when compared to the binary tree detectors generated by the NSA in [4]. The comparative results show that when ℓ and r are sufficiently large, the detector storage complexity and the detection time of PNSA are significantly smaller than NSA in [4] (36% and 50% less).

S	ℓ	r	Mem (%)	Time (%)
1000	50	12	0	0
2000	30	15	2.5	5
2000	40	17	25.9	42.7
2000	50	20	36.3	50

Fig. 6: Comparison of memory and detection time reductions.

We have conducted another experiment by choosing $\ell = 40$, $|S| = 20{,}000$ (S is the set of randomly generated binary strings of length ℓ) and varying r (from 15 to 40). Then, $\ell - r + 1$ trees were created using single NSA and other $\ell - r + 1$ compact trees were created using PNSA. Next, both detector sets were used to detect every $s \in S$. Figure 7 depicts the detection time of PNSA and NSA in the experiment. The results show that PNSA detection time is significantly smaller than that of NSA. For instance, when r is from 20 to 34, detection in PNSA is about 4.46 times faster than that of NSA.

Next experiment is conducted on Netflow dataset, a conversion of Tcpdump from well-known DARPA dataset to Netflow [22]. It

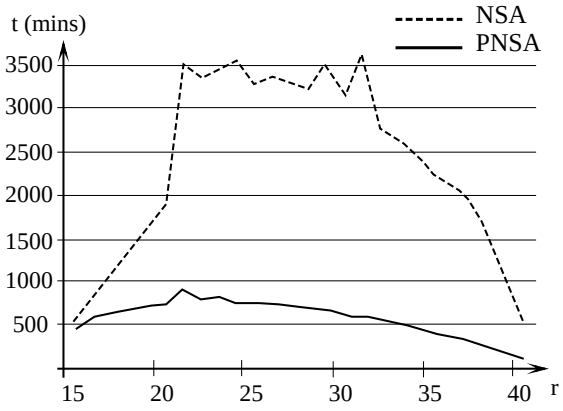


Fig. 7: Detection time of NSA and PNSA.

contains all 129,571 traffics (including attacks) to and from victims. Each flow in the dataset has 10 fields: Source IP, Destination IP, Source Port, Destination Port, Packets, Octets, Start Time, End Time, Flags, and Proto. All attacks, 22,104 flows, are labeled with text labels, such as neptune, portsweep, ftpwrite etc. We use all 107,467 normal flows as self samples. This self set is first converted to binary string of length 104, then we run our algorithm on r changing from 3 to 45. Figure 8 shows some of the experiment steps. The percentage of node reduction is in the final column. Figure 9 depicts the reduction of nodes in trees created by PNSA comparison to that of NSA for all r = 3,.., 45. It shows that the reduction is more than one third when the matching threshold greater than 19.

r	NSA	PNSA	Reduc.(%)
5	727	706	2.89
10	33,461	31,609	5.53
15	1,342,517	1,154,427	14.01
20	9,428,132	6,157,766	34.68
25	18,997,102	11,298,739	40.52
30	29,668,240	17,080,784	42.42
35	42,596,987	24,072,401	43.48
40	58,546,497	32,841,794	43.90
45	79,034,135	44,194,012	44.08

Fig. 8: Comparison of nodes generation on Netflow dataset.

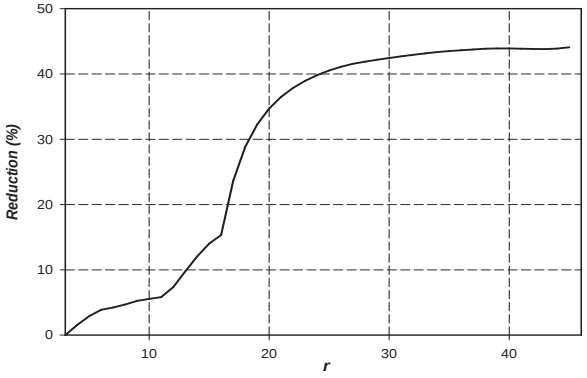


Fig. 9: Nodes reduction on trees created by PNSA on Netflow dataset.

The final experiment is on Spambase dataset [23], that consists of 4601 instances of ham and spam e-mail messages with 39.4% being spam. We use 2509 (90%) ham emails for training with each being converted to binary string of length 466. Figure 10 shows nodes reduction percentages of PNSA in comparison to PSA and NSA for all r = 4,.., 20. As an overall trend when r raises, it is clear that the number of reduced nodes goes down and goes up for PSA and NSA, respectively.

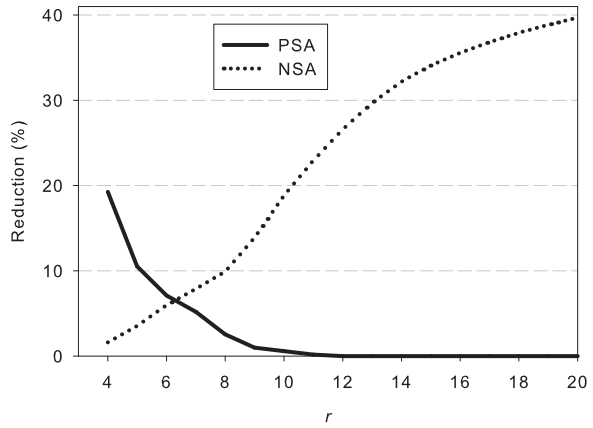


Fig. 10: Comparison of nodes reduction on Spambase dataset.

5. Conclusions

In this paper, we have proposed a novel approach to combining positive and negative selection algorithms. The new algorithm, PNSA, uses compact representation of the detector set as both positive and negative binary trees. PNSA was theoretically demonstrated to achieve better detector storage complexity in comparison to single NSA or PSA while maintaining the detection coverage, detector generation time complexity, and worst-case detection time complexity. This could potentially lead to lower detection time on real cases (i.e smaller average detection time complexity) as preliminarily confirmed in our experiments.

In near future, we are planning to test our algorithm on more real-world problems and other data sets such as virus detection, spam filtering, network attack identification (e.g. KDD CUP'99 data set). More importantly, we will mathematically formulate the average detection time complexity of PNSA and compare it with that of single NSA (or PSA) to theoretically prove the superior experimental results on detection time of PNSA obtained in this paper.

Acknowledgement

This work was partly funded by the National Foundation for Science and Technology Development (NAFOSTED) under grant 102.01-2010.09. The first author would like to thank Thai Nguyen University of Education for providing research facilities while doing this work.

References

[1] T. Stibor, J. Timmis, C. Eckert, A comparative study of real-valued negative selection to statistical anomaly detection techniques, Lecture notes in Computer science 3627 (2005) 262–275.

[2] D. Dasgupta, Artificial Immune Systems and Their Applications, Springer-Verlag, Berlin Heidelberg, 1998.

[3] Z. Ji, D. Dasgupta, Revisiting negative selection algorithms, Evolutionary Computation 15 (2007) 223–251.

[4] M. Elberfeld, J. Textor, Efficient algorithms for string-based negative selection, in: International Conference on Artificial Immune Systems, 2009, pp. 109–121.

[5] V. T. Nguyen, X. H. Nguyen, C. M. Luong, A novel combination of negative and positive selection in artificial immune systems, in: Proceedings of IEEE International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2013, pp. 6–11.

[6] A. S. A. Aziz, M. Salama, A. ella Hassanien, S. E. O. Harafi, Detectors generation using genetic algorithm for a negative selection inspired anomaly network intrusion detection system, in: Proceedings of the FedCSIS'2012, 2012, pp. 597–602.

[7] Z. Ji, Negative selection algorithms: from the thymus to v-detector, Ph.D. thesis, The University of Memphis (August 2006).

[8] S. Forrest, B. Javornik, R. E. Smith, A. S. Perelson, Using genetic algorithms to explore pattern recognition in the immune system, Evolutionary Computation 1 (1993) 191–211.

[9] S. Afaneha, R. A. Zitarb, A. A. Hamamic, Virus detection using clonal selection algorithm with genetic algorithm (VDC algorithm), Applied Soft Computing 13 (2013) 239–246.

[10] S. Forrest, S. A. Hofmeyr, A. Somayaji, T. A. Longstaff, A sense of self for UNIX processes, in: IEEE Symposium on Research in Security and Privacy, 1996, pp. 120–128.

[11] D. Dasgupta, S. Forrest, Novelty detection in time series data using ideas from immunology, in: Proceedings of the 5th International Conf. on Intelligent Systems, 1996.

[12] R. Murugesan, V. N. Kumar, A fast algorithm for solving jssp, European Journal of Scientific Research 64 (2011) 579–586.

[13] G. C. Silva, R. M. Palhares, W. M. Caminhas, Immune inspired fault detection and diagnosis: A fuzzy-based approach of the negative selection algorithm and participatory clustering, Expert Systems with Applications 39 (2012) 12474–12486.

[14] K. B. Sim, D. W. Lee, Modeling of positive selection for the development of a computer immune system and a self-recognition algorithm, International Journal of Control, Automation, and Systems 1 (2003) 453–458.

[15] Z. Fuyong, Q. Deyu, Run-time malware detection based on positive selection, Journal in Computer Virology 7 (2011) 267–277.

[16] Z. Fuyong, Q. Deyu, A positive selection algorithm for classification, Journal Computational Information Systems 7 (2012) 207–215.

[17] F. González, D. Dasgupta, J. Gómez, The effect of binary matching rules in negative selection, in: Proceedings of Genetic and Evolutionary Computation Conference (GECCO), 2003, pp. 195–206.

[18] J. Balthrop, F. Esponda, S. Forrest, M. Glickman, Coverage and generalization in an artificial immune system, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 2002, pp. 3–10.

[19] D. Dasgupta, F. Nino, A comparison of negative and positive selection algorithms in novel pattern detection, in: International Conference on Systems, Man, and Cybernetics, 2000, pp. 125–130.

[20] D. Dasgupta, S. Forrest, An anomaly detection algorithm inspired by the immune system, in: D. Dasgupta (Ed.), Artificial Immune Systems and Their Applications, Springer Berlin Heidelberg, 1999, pp. 262–277.

[21] F. Esponda, S. Forrest, P. Helman, A formal framework for positive and negative detection schemes, in: IEEE transactions on Systems, Man, and Cybernetics Society, 2004, pp. 357–373.

[22] Q. A. Tran, F. Jiang, J. Hu, A real-time netFlow-based intrusion detection system with improved BBNN and high-frequency field programmable gate arrays, in: Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, IEEE Computer Society, Los Alamitos, CA, USA, 2012, pp. 201–208.

[23] K. Bache, M. Lichman, UCI machine learning repository (2013).
URL `http://archive.ics.uci.edu/ml`