Detector generation for artificial immune system in a dynamic environment

Duong Thuy Huong, Ngo Thi Tu Quyen, Le Bich Lien, Nguyen Van Truong

Abstract— **The artificial immune system (AIS) is an approach of bioinformatics, which is the concept of artificial intelligence systems, problem-solving based on the principles, functions and operating models of the human immunity system. AIS has been recently applied to computers security. One of the important features in such systems is that the data is constantly changing, and there is a constant change back and forth between the normal and abnormal codes. This has led to the construction of an artificial immune system that is effective in dynamic environments and for large data. We propose an immune algorithm that can work effectively in term of running time in dynamic environment.**

Index Terms—**Detector, intrusion, immune system.**

I. Introduction

Following the traditional approach, AIS is considered to be an effective method to protect static data sets [1]. That is, the data to be protected is a fixed set over time. And this also reflects the exact mechanism of action of the biological immune system: the self types of the normal biological body are always constant. But if you carefully analyze the security problem, you will see many unreasonable situations. A typical example is: A website address which is considered healthy, but may be hijacked by bad hackers and changed content with bad content. After a while, the website owner regains control (for example, through a domain name provider, through a server provider, etc.), and the internal website returns to its original form. Thus, if the website address is a data element to be protected, it can be likened to a body self or a foreign self depending on the particular circumstances. Therefore, it is important for AIS to be applicable in dynamic environments.

In this paper, Artificial Immune System (AIS), a multidisciplinary research area that combines the principles of immunology and computation, is used for experiments on the proposed representation.

AIS is inspired by the observation of the behaviors and the interaction of normal component of biological systems - the self -and abnormal ones - the nonself. Negative Selection Algorithm (NSA) is a popular model of AIS mainly designed for one-class learning problems.

Duong Thuy Huong, University of Information Technology and Communications, Thai Nguyen, Thai Nguyen City, Vietnam.

Ngo Thi Tu Quyen, Faculty of Mathematics, Thai Nguyen University of Education, Thai Nguyen City, Vietnam.

Le Bich Lien, Faculty of Mathematics, Thai Nguyen University of Education, Thai Nguyen City, Vietnam.

Nguyen Van Truong, Faculty of Mathematics, Thai Nguyen University of Education, Thai Nguyen City, Vietnam.

Given a collection of self patterns S, a typical NSA comprises of two phases: detector generation and detection [5]. In the detector generation phase (Fig. 1.a), the detector candidates are generated randomly and censored by matching them against given self samples taken from the set S. The candidates that match any element of S are eliminated and the rest are kept and stored in the set D. In the detection phase (Fig. 1.b), the collection of detectors are used to distinguish self (system components) from nonself (outlier like viruses, worms, etc.). If an incoming data instance matches any detector, it is claimed as nonself, and it is claimed as self otherwise.

From a machine learning perspective, negative selection is usually described as an anomaly detection technique. Since its introduction, NSA has been a source of inspiration for many computing applications, especially for intrusion detection [6], [7].

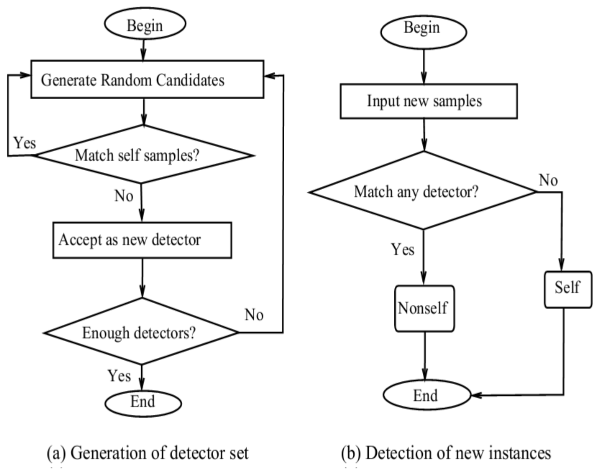

Fig. 1. Outline of a typical negative selection algorithm [13].

With respect to binary-based AIS using discrete detector set, to the best of our knowledge, the only algorithm for generating a perfect and discrete set of detectors was proposed by T. Stibor in [11] and by S. T. Wierzchoń in [14].

In the below figure, we illustrate the process for generating a detector set by using the number of five contiguous matches required for a match. The string to be protected is logically segmented into five equal-length “self” strings. To generate the detector, random strings are produced and matched against each of the self strings. The first two strings, 00000111 and 00000010, are eliminated because they both match self string 00000110 at at least five contiguous positions. The string 11101001 fails to match any string in the self at at least five contiguous positions, so it is accepted into the detector set.

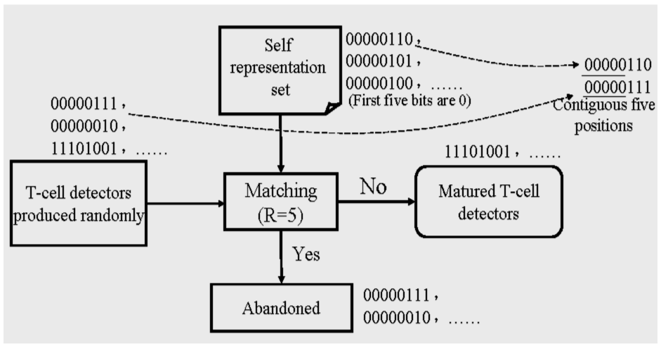

Figure 2. Generation of Valid Detector Set

Next section presents algorithms to update detector set when ever self elements delete from or add to self set.

II. Detector Generation Algorithms in a Dynamic Environment

A. *Some notations*

S: a set of self strings (in binary form) of equal length (= L) of data to be protected.

R: number of consecutive bit positions to match.

Match: Match(a, b) = true if 2 strings a, b match at least r consecutive positions.

Detector(d): a string that does not match any data series to be protected.

Detector set (D): a set of detectors.

Self set (S): a set of self

N (Nonself): a strange bit string appearing in S

Holes: strange strings appearing in S that detectors cannot detect.

B. *Classical detector generation algorithm*

Input: The data set needs to be protected.

Output: Positioning matrix A and detector set D

From the initial input data set, we create a set of S selfs including s selfs that are binary strings with a fixed length of L, with each generated s self, we construct a locating matrix A Matrix A has size $2^R * (L - R + 1)$ (with line index from 0 -> $2^R - 1$, and column index from 1 -> L - R +1), initialized to original value beginning with 0. Matrix construction algorithm A is shown below. After creating matrix A we use the algorithm to generate the detector.

Algorithm for building a positioning matrix A is described as follows:

Initially initialize A[i, j] = 0 with i = 0 .. (2^R -1) and j = 1 .. (L-R + 1)

From the initial data set, we create bit sequences of fixed length L

Each time the bit stream s is generated

```
For j ∈ [1, L-R + 1] do
    Begin
        s1 = copy (s, j, R);
        i = decimal integer of the string s1;
        A[i, j] = A[i, j] +1;
  End;
```

Algorithm to generate all detectors is as follows.

Denote nd is the number of detectors, de is the array of probes, t is the array of intermediate strings, and dt is the number of elements of the array t.

Algorithm initialization array D:

```
nd = 0;
For i ∈ [0, $2^R$ -1] do
If A[i,1] = 0 then
Begin
   Inc(nd);{this means increase nd by 1}
   De(nd) = string of length R bits has a decimal value of i;
End ;
For i ∈ [2 ,L - R + 1] do
Begin
   dt = 0;
   For j ∈ [1,nd] do
   Begin
          s1 = copy(de[j],i,R-1);
          d = decimal integer of the string (s1+ “0”) ;
          If A[d, i] = 0 Then
               Begin
                     Inc(dt);
                     t[dt] = de[j] + “0”;
               End;
                  d = decimal integer of the string (s1+ “1”);
           If A(d, i) = 0 Then
                  Begin
                       Inc(dt);
                       t[d] = de[j] + “1”;
                  End;
                  nd = dt;
           For k ∈ [1,nd] do
                       de[k] = t[k];
           End;
End;
```

Suppose we have set S of the series of bit bits of length L = 5 and R = 3, S = {s1 = 01011, s2 = 11001, s3 = 10110, s4 = 00101, s5 = 01010}. Then set D consists of 6 detectors: D = {d1 = 00000, d2 = 01100, d3 = 01111, d4 = 10000, d5 = 11110, d6 = 11111}.

C. *Detector generation algorithm in case self are added to self set S*

Suppose we have created a localization matrix A and a set of D-detectors in the absence of an increase in the S-self set. When the S self set increases, the number of detectors in the D detector set can be reduced or remained the same. And we can conduct a new D detector set for the new self set after increasing as the algorithm is presented in section B. However, this approach is not really effective because we have to recreate the new positioning matrix A and thereby build a new set of D detector. Here we propose a faster generation algorithm and take advantage of the matrix A and set D detector has created.

Input: Set S1 is the set of self added to S

The A-locating matrix and the set of D selfs are created corresponding to the initial set of S selfs

Output: Positioning matrix A and corresponding detector set D after addition;

For each s ∈s1, we update the matrix A according to the matrix construction algorithm presented. During this update process, if encountering a position where A[i, j] is 0 then changes to 1, then we proceed to create a new set of detectors D1 at this position. We use two procedures to create the array of tuners that go to the left (from column j position to 1) called LeftDetec and the array when going to the right (from column j position to L-R + 1) called RightDetec. These two procedures will be presented below. Then proceed to enroll the detector set D1 and remove d ∈ D1 if it is in D.

The algorithm description is as follows:

For every s ∈S1 do

```
For j ∈ [1, L-R + 1] do
Begin
   s1 = copy (s, j, R);
   i = decimal integer of the string s1;
if A[i, j] = 0 then
begin
   nd1 = 1;
   nd2 = 1;
   de1[1] = a binary string of length R with a decimal value of i;
      de2[1] = de1[1];
      LeftDetec(nd1, j);
      RightDetec(nd2, j);
       nd3 = 0;
       For i = 1 To nd1 do
       For j = 1 To nd2 do
       Begin
            nd3 = nd3 + 1;
            de3[nd3] = de1[i] + Copy(de2[j], R + 1, length(de2[j]));
   end;
Open the file containing detector D set to read the string array with the number of nd elements;
Delete this file;
n = nd;
For j = 1 To nd do
If de3[i] = de[j] Then
Begin
   n = n - 1;
   de[j] = "";
 End;
Open the file to write data:
   Line 1 contains the number n;
   The next line contains the string de[i] with i ∈ [1, n];
End;
End;
```

Denote t, de1 are binary string arrays and dt, nd1 is the number of corresponding elements (t is used as an intermediate variable), respectively.

+, The LeftDetec procedure creates the array of detectors to the left

```
LeftDetec Procedure (var nd1 As Integer; vtri As Integer);
For i = (vtri - 1) downto 1 do
Begin
   dt = 0;
   For j = 1 To nd1 do
   begin
      s1 = copy (de1[j], 1, R - 1);
      d = decimal integer of the string (s1 + "0");
       If A[d, i] = 0 Then
      Begin
             dt = dt + 1;
             t[dt] = "0" + de1[j];
      End;
      d = d + $2^{(R-1)}$;
      If A[d, i] = 0 Then
      Begin
            dt = dt + 1;
            t[dt] = "1" + de1[j];
      End;
  End;
        nd1 = dt;
        For k = 1 To nd1 do
            de1[k] = t[k];
end;
end;
```

+, The RightDetec procedure creates the detector array going to the right

```
RightDetec Procedure (var nd1 As Integer; vtri As Integer);
Begin
h = 1;
For i ∈ [2, L - R + 1] do
Begin
dt = 0;
   For j∈ [1, nd1] do
  Begin
Inc (h);
     s1 = copy (de (j), h, R-1) ;
      d = decimal integer of the string (s1 + "0");
      If A[d, i] = 0 Then
         Inc (dt);
          t[d] = de[j] + "0";
 End;
 d = decimal integer of the string (s1 + "1");
 If A[d, i] = 0 Then
          begin
               Inc (dt);
               t[d] = de[j] + "1";
          End;
           nd = dt;
For k ∈ [1, nd] do
               de[k] = t[k];
End;
End;
End;
```

D. Detector generation algorithm in case self are deleted from self set S

Similar to the case of increased S self set, in the case of reduced S self set we also have a detector generation algorithm. Note that in this case the set of probes can be increased or remained constant, depending on the position A[i, j] changes the state from 1 to 0;

Input: Set S1 is the set of selfs to be deleted in the original S set

The A-locating matrix and the set of D selfs are created corresponding to the initial set of S selfs

Output: Positioning matrix A and corresponding detector set D after deletion;

In this algorithm, we will meet the state A[i, j] whose value is greater than 0 returns to 0, if encountering such a position, we proceed to create a new set of probes D1 in position. this position. Then add set D1 to the original set D. The algorithm description is as follows:

```
For every s ∈ S1 do
For j ∈ [1, L-R + 1] do
Begin
A[i, j] = A[i, j] - 1;
if A[i, j] = 0 then
begin
   s1 = copy (s, j, R);
   i = decimal integer of the string s1;
   nd1 = 1;
   nd2 = 1;
   de1[1] = a binary string of length R with a decimal value of i;
     de2[1] = de1[1];
     LeftDetec(nd1, j);
```

```
    RightDetec(nd2, j);
    nd3 = 0;
    For i = 1 To nd1
     For j = 1 To nd2
     Begin
       nd3 = nd3 + 1;
       de3[nd3] = de1[i] + Copy(de2[j], R + 1, length(de2[j]);
    End;
Open the file containing detector D set to read the string array
with the number of nd elements;
Delete this file;
For j ∈ [nd, nd + nd3] do
       De[j] = de3[i-nd];
Open the file to write data:
   Line 1 contains the number nd + nd3;
   The following lines write the string de[i] with i ∈
   [1,nd+nd3];
End;
End;
```

III. Discussions

In the process of creating a detector, in fact we do not have to save the self set but only the positioning matrix A because when creating the set of detector D we only rely on matrix A. Because the size of table A depends on the length L and the number of neighboring bits R. So we can choose L and R reasonable to store on internal memory (RAM). Saving matrix A is much less memory-intensive than saving self set S of bit sequences of fixed length L, especially when the input data is large.

Because A is a 2-dimensional table, stored on internal memory, access to element A[i, j] takes only O(1). So the computational speed will increase much more than having to read the self strings from external memory. So even if the protection data has changed, very large, we still only need table A with 2^R rows, (L-R + 1) column is enough.

In the case of increasing S self we only handle A[i, j] positions with transition state from 0 to 1, so we can also use the method when encountering the position where A[i , j] with the above properties, we do not immediately handle but re-mark, once we have read all the added self sets, we review the matrix and handle the highlighted selfs. Alternatively, we convert i to a binary bit sequence of length R and then compare it to the string of R bits removed from position j of each d in D, if equal then we remove this detector from D However, both of these methods are not as efficient as the method described above, especially in the case of a large D detector set, because accessing D now takes a lot of time and takes up storage space for string array in comparision with creating the new set of detectors.

In fact, in matrix A there are locations with a value of 0 but not involved in the process of creating a detector. So when we meet such positions, how will we handle them effectively? By assuming in these selfs we can create a new set of detectors called D1, and we can easily prove that $D1 \cap D = \varnothing$. Hence D - D1 = D, that is, D remains unchanged.

From the above analysis, we can find the solution: From the set of detector D, we construct the A_D matrix to build the positioning matrix A with the input set D whenever $d \in D$ is created with the following change:

```
   For j ∈ [1, L-R + 1] do
   Begin
      s1 = copy (s, j, R);
      i = decimal integer of the string s1;
      $A_D[i, j] = -1$;
   End;
```

And at selfs A[i, j] ≠ 0 then we assign $A_D[i, j] = 1$, then the positions $A_D[i, j] = 0$ are the positions not participating in the detector generation process. So in the process of creating a new detector when encountering state A[i, j] changes from 1 to 0, we check if $A_D[i, j] \neq 0$ then proceed to create a detector and simultaneously update the A_D matrix.

IV. Conclusions

This article has provided an overview of AIS and propose a new algorithm in the field. The algorithm is simulated in dynamic environment effectively in term of running time.

In the future, we will develop this research in the following directions: 1-Improve, upgrade illustration program so that the program can be applied in the network environment; 2-combine with advanced algorithms of AIS such as: AiNet, RAIN, Clone; 3 - Improve algorithms to handle in case the existence of holes self.

Acknowledgement

This research is based upon work supported in part by Thai Nguyen University for university's research, code number DH2017-TN01-03.

References

.
[1]. L. N de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Intlligence Approach*, Springer-Verlag, 2002.
[2]. M. Elberfeld and J. Textor. Efficient algorithms for string-based negative selection. In International Conference on Artificial Immune Systems, pp. 109–121, 2009.
[3]. M. Elberfeld and J. Textor. Negative selection algorithms on strings with efficient training and linear-time classification. Theoretical Computer Science, 412(6):534 – 542, 2011.
[4]. F. Gonzalez, D. Dasgupta, J. Gomez, "The effect of binary matching rules in negative selection", in: Genetic and Evolutionary Computation Conference (GECCO), pp. 195–206, 2003.
[5]. Z. Ji. Negative Selection Algorithms: from the Thymus to V-detector. PhD thesis, The University of Memphis, August 2006.
[6]. Z. Ji, D. Dasgupta, "Revisiting negative selection algorithms", Evolutionary Computation, vol. 15, no. 2, pp. 223–251, 2007.
[7]. J. Kim, P. J. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross. Immune system approaches to intrusion detection - a Review. Natural Computing, 6:413–466, Dec. 2007.
[8]. L. X. Peng, Y. F. Chen, "Positive selection-inspired anomaly detection model with artificial immune", in: International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 56–59, 2014.
[9]. P. H. Pisani, A. C. Lorena, A. C. Carvalho, "Adaptive positive selection for keystroke dynamics", J. Intell. Robotics Syst., vol. 80, no. 1, pp. 277–293, 2015.
[10]. 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, vol. 1, no. 4, pp. 453–458, 2003.
[11]. T. Stibor, K. M. Bayarou, and C. Eckert, "An investigation of R-chunk detector generation on higher alphabets," in Genetic and Evolutionary Computation Conference (GECCO), vol. 3102 of Lecture Notes in Computer Science, pp. 299–307, 2004.
[12]. D. Y. Yeung, Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models", Pattern Recognition, vol. 36, no. 1, pp. 229–243, 2003.
[13]. H. Yang, T. Li, X. Hu, F. Wang, Y. Zou, "A survey of artificial immune system based intrusion detection", The Scientific World Journal, 2014.

[14]. S. T. Wierzchoń. Generating optimal repertoire of antibody strings in an artificial immune system. In IIS'2000 Symposium on Intelligent Information Systems, pp. 119–133, 2000.

[15]. W. Zheng at el, A Rapid r-continuous Bits Matching Algorithm for Large-scale Immunocomputing, in *Proceedings of the International Conference on Computer Science and Software Engineering*, 431- 434, 2008.

BIOGRAPHY

Duong Thuy Huong is a lecturer in the Faculty of Information Technology - University of Information Technology and Communications, Thai Nguyen. She finished her master course on Computer science at Thai Nguyen University in 2014. She has taught a wide variety of courses for UG students and guided several projects.

Ngo Thi Thu Quyen is a lecturer in the Faculty of Mathematics at Thai Nguyen University of Education, from where she received a Bachelor of Informatics in 2000. She is a Ph.D. at Thai Nguyen University of Education. She teaches many course for IT students. She has published several papers in national and international journals.

Le Bich Lien is a lecturer in the Faculty of Mathematics at Thai Nguyen University of Education, from where she received a Bachelor of Informatics in 2004. She finished her master course on Computer science at Thai Nguyen University of Information and Communication Technology in 2007. She has published several papers in national and international journals.

Nguyen Van Truong is a lecturer in the Faculty of Mathematics at Thai Nguyen University of Education. He has taught a wide variety of courses for UG students and guided several projects. He has published several papers in national and international journals. His research interests are embedded systems and artificial immune systems.