Algorithms of Non-self Detector by Negative Selection Principle in Artificial Immune System*

Ying Tan and Zhenhe Guo

School of Information Science and Technology,
University of Science and Technology of China,
Hefei 230027, P.R. China
ytan@ustc.edu.cn

Abstract. According to the principles of non-self detection and negative selection in natural immune system, two generating algorithms of detector are proposed in this paper after reviewing current detector generating algorithms used in artificial immune systems. We call them as Bit Mutation Growth Detector Generating Algorithm (BMGDGA) and Arithmetical-compliment Growth Detector Generating Algorithm (AGDGA) based on their operational features. The principle and work procedure of the two detector generating algorithms are elaborated in details in the paper. For evaluation of the proposed algorithms, they are tested and verified by using different datasets, and compared to Exhaustive Detector Generating Algorithm (EDGA). It turns out that the proposed two algorithms are superior to EDGA in detection performance and computational complexities.

1 Introduction

The immune system defends the body against harmful diseases and infections and plays very important adjustment functions in the whole lifetime of biological creatures. It has many features that are desirable for information security research and problem solving of complex problems encountered in many engineering fields. Based on natural immune principles, many algorithms and architectures are proposed [1]-[7] in many engineering fields. Among them, the non-self recognition interests greatly IT and computer researchers and stimulates a lot of artificial immune systems (AIS) for change detection and pattern recognition. Forrest, et.al., [7] proposed the well-known negative selection algorithm (NSA) as a detector generating algorithm by simulating the negative selection process of T cells generating in thymus. This paper studies the detector-generating algorithm carefully, and proposes two-detector generating algorithms based-on non-self detection and negative selection principles from the perspective of optimizing detector generation. Two algorithms are elaborately described in

* This work was supported by the Natural Science Foundation of China with Grant No. 60273100.

L. Wang, K. Chen, and Y.S. Ong (Eds.): ICNC 2005, LNCS 3611, pp. 867 – 875, 2005.

theory and implementation and completely compared with Exhaustive Detector Generating Algorithm (EDGA).

2 Negative Selection Principle and Related Algorithms

2.1 Negative Selection Principle and Algorithm

Negative selection is one of important phases of generation and maturation of T cells in thymus. T-cells with essentially random receptors are generated in thymus. Before they are released to the rest of body, those T-cells that match self are deleted. This process is called negative selection process based on which a number of detector generating algorithms are created in many artificial immune systems (AIS). In 1994, Forrest, et.al, proposed so-called negative selection algorithm (NSA) inspired by negative selection principle (NSP) [7]. NSA consists of two stages of *censoring* and *monitoring*. The censoring phase caters for the generation of change-detectors. Subsequently, the system being protected is monitored for changes using the detector set generated in the censoring stage.

2.2 Current Detector Generating Algorithms

There are a number of detector-generating algorithms with different matching rules applied for 'self' and 'non-self' matching methods. Currently, for binary code, there are mainly three kinds of matching rules, which are perfect matching, r-contiguous bits matching [2] and Hamming distance matching. Exhaustive detector generating algorithm (EDGA) is suitable for three matching rules and is used to repeat the NSP till the number of detectors meets a preset demand or candidate set is vanished. However, negative selection algorithm with mutation [9] has different evaluating rules in the negative selection process from EDGA. With r-contiguous bits matching rule, the related algorithms mainly include Liner Time Detector Generating Algorithm and Greedy Detectors Generating Algorithm [9][10].

3 Bit Mutation and Arithmetic-Compliment Growth Algorithms

3.1 Growth Algorithm

The main difference between growth algorithm and NSA is the generating method of candidate detectors. For NSA, each detector candidate in EDGA is randomly selected from whole candidate space. Thus, after the number of detector candidate space N_{r0} is determined, EDGA randomly selects N_{r0} detectors to construct the detector candidate set R_0. Then, the algorithm generates detector set R through negative selection process. On the other hand, growth algorithm does not need to maintain a huge detector candidate set R_0. It directly generates the detector set R by utilizing detector mutation or growth in whole shape space and combining with negative selection process. Its flow chart is shown in Fig.1.

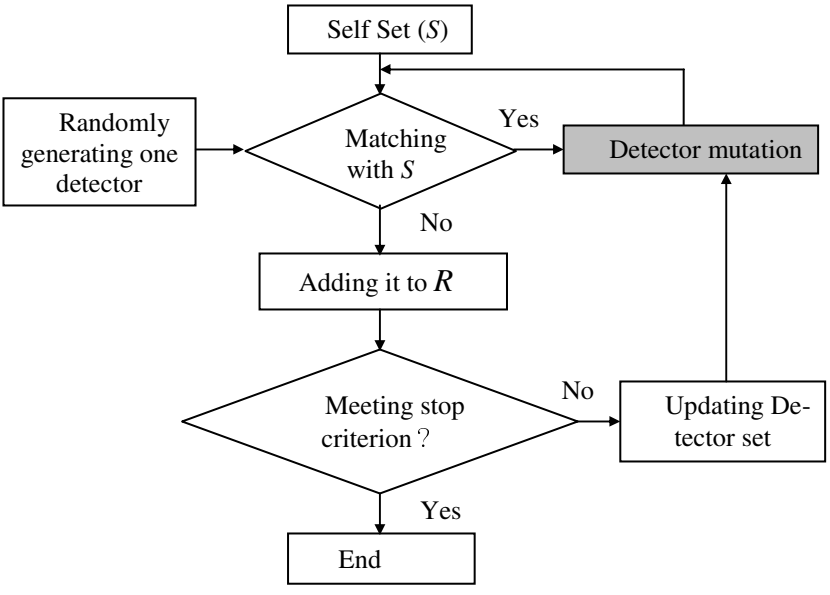

Fig. 1. Flow chart of detector generation growth algorithm

Growth Algorithm

Step 1. Generating self set S with its number N_s.

Step 2. Generating one detector generating seed which is randomly selected from the whole shape space.

Step 3. Matching the new detector candidate with S.

Step 4. Experiencing a negative selection process. If the candidate is not matched with S, then a new one is generated and added into R.

Step 5. If the stop criterion is met, then exit.

Step 6. Mutating the candidate and going to step 3.

3.2 Bit Mutation and Arithmetical-Compliment Growth Methods

Here, we design two detector mutation methods of bit mutation and arithmetical-compliment growth according to different detector mutation rules.

3.2.1 Bits Mutation

This method is similar to bit mutation of Genetic Algorithm. But this algorithm mutates multiple bits of detector candidate, not only one bit. If we let the string length be l and the maximum number of mutation bits be N_m ($N_m = m$), then, at one time, the mutated bits of detector candidate is less than or equal to m.

Bit Mutation Algorithm

Step 1. Set the maximum number of mutation bits N_m.

Step 2. Input detector DetectorM to be mutated.

Step 3. Generate mutation seed MutationBits (its length is same as detector): N_m bits of MutationBits are randomly generated at range of 1 and l and set to 1, and others are set to zero.

Setp 4. Detector mutation: an exclusive OR operation performed on the corresponding bits of arrays DetectorM and MutationBits.

Pascal language description of bits mutation algorithm:

```
Program Bits_Mutation(DetectorM, N_m )
    var
     MutationBits    : Longword;        //Mutation seed
     MutationBit     : array[1.. N_m ] of Byte;
    begin
       MutationBits := 0;
       For i := 1 to N_m do              //Generating mutation seed
       begin
            MutationBit[i] := Random( l );
            MutationBits := (1 shl MutationBit[i]) or MutationBits;
       end;
       DetectorM:= DetectorM xor MutationBits; //Detector mutation
    end;
```

3.2.2 Arithmetic-Compliment Growth Algorithm

This mutation method consists of two phases. Phase 1 is used to generate mutation seed and phase 2 is to mutate the detector candidate. The process of generating mutation seed in phase 1 is same as bit mutation method described above except that the sum of arithmetic-compliment of mutation seed and detector mutation is used as the detector candidate.

Pascal language description of Arithmetic-compliment growth algorithm:

```
Program Arithmetical-compliment_growth(DetectorM, N_am )
    var
      MutationO     :Longword;              //Mutation seed
      MutationBit   :array[1.. N_am ] of Byte;
    begin
      MutationO := 0;
      for i := 1 to N_am do                 //Generating mutation seed
      begin
     MutationBit[i] := Random( l );
     MutationO := (1 shl MutationBit[i]) or MutationO;
      end;
      DetectorM := MutationO + DetectorM;
      If DetectorM >= 2^l then DetectorM := DetectorM - 2^l ;
    end;
```

3.3 BMDGA and AGDGA

Once the proposed two mutation methods are separately applied to the detector mutation part of the growth algorithm in Fig.1, we can obtain two kinds of detector generating algorithms which are called as bit mutation algorithm (BMDGA) and arithmetical-complement growth algorithm (AGDGA), respectively.

4 Experiments

4.1 Experimental Goals

There are three main goals for our experiments. They are:

- Validating algorithms' quality of generated detector set by using detection rate P_c.
- Comparing the algorithm's complexity to EDGA.
- Setting parameters including N_m and N_{am}, according to algorithms' performance.

4.2 Selection of Experimental Parameters

We set two kinds of experiments. One chooses random dataset with string length 8, 16 and 24 bits, respectively. Another experiment is to detect the changes of static files.

4.3 Random Dataset Experiments

4.3.1 8-Bit Dataset Experiments

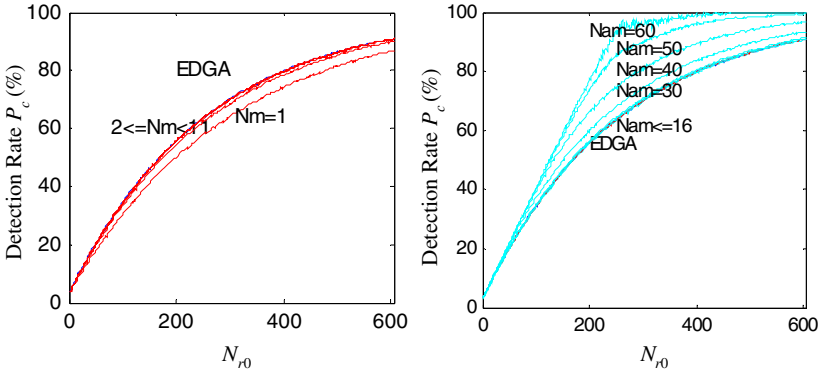

Fig. 2. Experimental results of Detection rate P_c versus N_{r0} when N_s is constant, where N_s =8, N_{r0} increases from 1 to 604, size of test set is 256, and 1000 runs. (a) Curves of P_c versus N_{r0} for EDGA and BMGDGA, (b) Curves of P_c versus N_{r0} for EDGA and AGDGA.

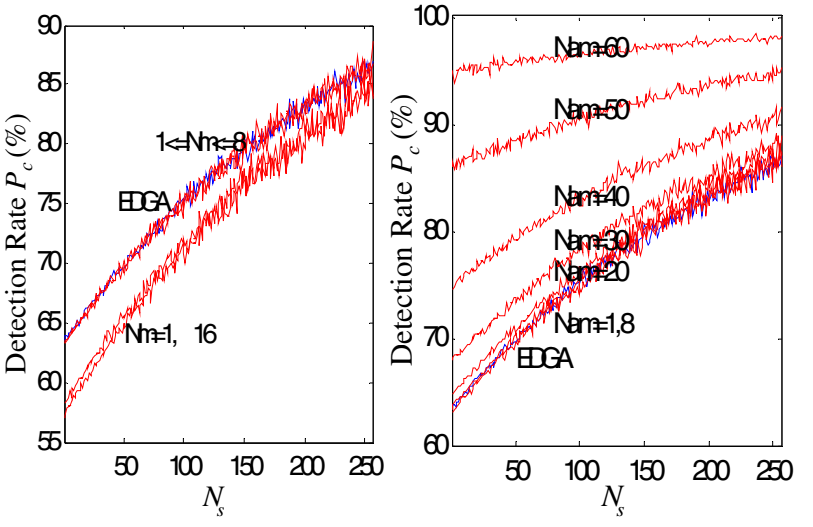

Fig. 3. Experimental results when N_{r0} is constant and N_s changing from 1 to 256. (a) When N_{r0} =256, the experimental result comparison of EDGA to BMGDGA, (b) When N_{r0} =256, the experimental result comparison of EDGA to AGDGA.

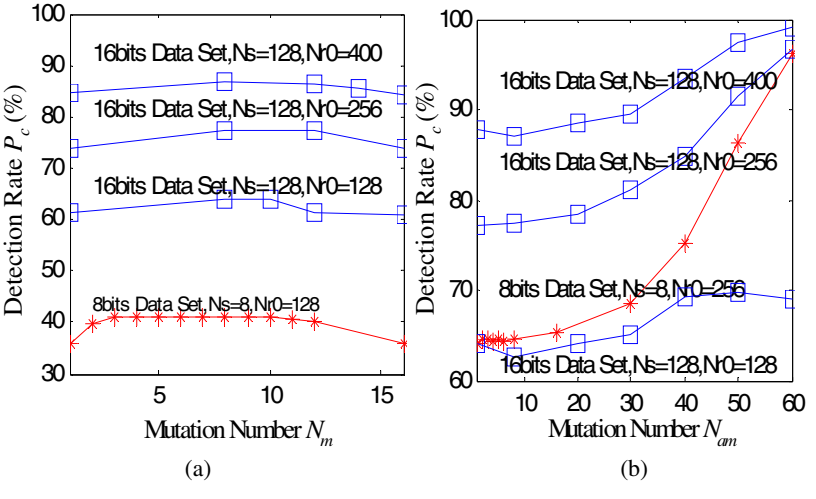

Fig. 4. Experimental results of detection rate versus mutation number, where N_s, N_{r0} are constants. To different datasets, N_m is increasing from 1 to 16, and N_{am} is increasing from 1 to 64. (a) Curve of P_c vs. N_m, (b) Curves of P_c vs. N_{am}.

4.3.2 16-Bit and 24-Bit Dataset Experiments

a. 16-bit dataset experiment

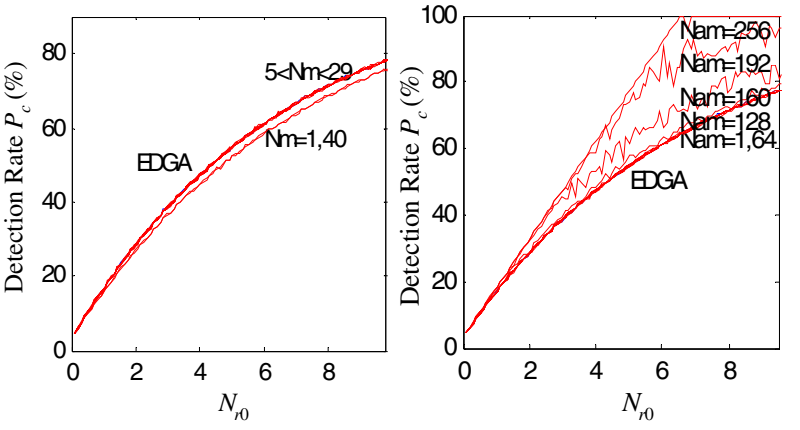

Fig. 5. 16-bit dataset experimental results when N_s is constant. (a) Experimental results of BMGDGA and EDGA, (b) Experimental results of AGDGA and EDGA.

Table 1. Computational complexity comparison for 16-bit dataset

Algorithm	Parameters setting	Computational time (ms)	Note
EDGA	N_{r0}=65536, N_S=2048	730	
BMGDGA	N_{r0}=65536, N_S=2048 N_m=8	594.5	10 runs
	N_{r0}=65536, N_S=2048 N_m=4	585	10 runs
AGDGA	N_{r0}=65536, N_S=2048 N_{am}=4	584.5	10 runs
	N_{r0}=65536, N_S=2048 N_{am}=200	909.5	10 runs

Notice: experimental platform: Intel Pentium 993M CPU, 256M memory and Windows Me.

b. 24-bit dataset experiment

Table 2. Computational complexity comparison for 24-bit dataset

Algorithm	Parameters setting		Detection rate	Computational time (ms)
EDGA	N_S=2048, N_{r0}=65536, N_t=65536		0.3787231445	731.5
BMGDGA	N_S=2048	N_m=1	0.3834533691	534.2
	N_{r0}=65536	N_m=16	0.3952026367	538
	N_t=65536	N_m=24	0.4139709472	547
AGDGA	N_S=2048	N_{am}=1	0.4036712646	534
	N_{r0}=65536	N_{am}=256	0.3980255126	899.1
	N_t=65536	N_{am}=512	0.4000701904	1252.66

4.3.3 Discussions

To BMGDGA, the detection rate P_c is not lower than EDGA's under all circumstances, and when $N_m = l/2$, the detection rate reaches the best. Its computational complexity is also superior to EDGA's, with almost same memory space. So the overall performance of proposed BMGDGA is better than EDGA.

To AGDGA, when $N_{am} \in [1, l^2]$, its detection rate P_c is all, superior to EDGA's. With the increasing of N_{am}, P_c and computational complexity are increasing. When N_{am} is less than l, its computational complexity is less than EDGA's. With the increasing of N_{am}, the computational complexity slowly exceeds EDGA's a bit. However, their space complexities are almost same. Therefore the overall performance of proposed AGDGA is also better than EDGA.

4.4 Change Detection of Static Files

We conduct three experiments with two algorithms to validate their detection abilities for the change of static files, and compare to EDGA. First of all, we compare two different files by using the algorithms. We select two files of 'FTP.exe' and 'Ping.exe' and define 'Ping.exe' as self (S). The experimental results of anomaly number are listed in the first sub-column of Anomaly number column of Table 3. Secondly, they are used to detect the change of programs. Here the protected program files are compiled in Delphi environment and some simple functions are added into the program files to form the changed program. The experimental results of anomaly number are listed in the second sub-column of Anomaly number column of Table 3. The third is anomaly detection of the file infected by a virus, where we define a benign file '*.exe' as protected file and detect its changing when infected by Fun Love virus. The experimental results of anomaly number are listed in the third sub-column of Anomaly number column of Table 3. Experimental results are all compared to EDGA and listed in Table 3 for convenience.

Table 3. Experimental results of anomaly numbers for file comparison (first sub-column of last column), program change detection (second sub-column of last column), and anomaly detection of file (third sub-column of last column).

Algorithms	Parameters setting		Anomaly number		
EDGA	Detector length l =16bits		3775.2	4138.5	469.3
BMGDGA	Detector length l =16bits	N_m =1	3389.5	4018.3	455.2
		N_m =10	3724.8	4551.8	467.1
		N_m =16	3709.6	4229.2	466.7
AGDGA	Detector length l =16bits	N_{am} =1	3718.2	4281.1	473.1
		N_{am} =128	4073.7	4429.5	483.4
		N_{am} =256	4540.8	5512.8	555.7

It can be seen from the experimental results that we can obtain the same results as random dataset experiments. To BMGDGA, when N_m=1, the quality of detector set generated by this algorithm is worst. When N_m=l/2~ l, the performance of BMGDGA is almost same to EDGA. To AGDGA, when N_{am}=1, its performance reaches EDGA. As N_{am} increases, the detection performance is also increasing. It turns out from our experiments that it is possible to detect the changes of static files with these two algorithms.

5 Conclusions

This paper proposes two novel detector-generating algorithms based on negative selection principle of immune system. Extensive experimental results show that they all have better performances than current EDGA. Under all circumstances, the AGDGA's detection rate is always higher than EDGA. As the increase of the number of mutation bits, the detection rate P_c increases and the computational complexity also increases. In summary, the proposed algorithms outperform EDGA in both detection performance and computational complexity.

References

1. D'haeseleer, P.: An Immunological Approach to Change Detection: Theoretical Results. Proceedings of the 9th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press (1996).
2. D'haeseleer, P., Forrest, S., Helman, P.: An Immunological Approach to Change Detection: Algorithms, Analysis and Implications. Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1996).
3. D'haeseleer, P.: A Change Detection Method Inspired by the Immune System: Theory, Algorithms and Techniques. Technical Report CS95-06, The University of New Mexico, Albuquerque, NM, 1995.
4. D'haeseleer, P.: Further Efficient Algorithms for Generating Antibody Strings. Technical Report CS95-03, The University of New Mexico, Albuquerque, NM, (1995).
5. Somayaji A., Hofmeyr S., Forrest S.: Principles of a Computer Immune System. 1997 New Security Paradigms Workshop (1998) 75-82.
6. Forrest S., Hofmeyr S., Somayaji A., Longstaff T.A.: A Sense of Self for Unix Processes. In Proceedings of 1996 IEEE Symposium on Computer Security and Privacy (1996).
7. Forrest S., Perelson A.S., Allen L., Cherukuri R.: Self-Nonself Discrimination in a Computer. In Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA: (1994).
8. Kim J., Bentley P.: Immune Memory in the Dynamic Clonal Selection Algorithm. In Proceedings of ICARIS'02, (2002).
9. Ayara M., Timmis J., de Lemos R., de Castro L., Duncan R.: Negative Selection: How to Generate Detectors. In Proceedings of ICARIS'02, (2002).
10. Singh S.: Anomaly Detection Using Negative Selection Based on the r-contiguous Matching Rule. In Proceedings of ICARIS'02, (2002).