Negative Selection: How to Generate Detectors

Modupe Ayara, Jon Timmis, Rogério de Lemos
Computing Laboratory
University of Kent at Canterbury,
CT2 7NF, U.K.
{moa2, j.timmis,
r.delemos}@ukc.ac.uk

Leandro N. de Castro

Department of Computer and Electrical Engineering
State University of Campinas, Brazil.
lnunes@dca.fee.unciamp.br

Ross Duncan

Advanced Technology and Research Group
Self Service Strategic Solutions
NCR FSG Ltd., U.K.
ross.duncan@scotland.ncr.com

Abstract

The immune system is a remarkable and complex natural system, which has been shown to be of interest to computer scientists and engineers alike. This paper reports an on-going investigation into the usefulness of the negative selection metaphor for immune inspired fault tolerance. Various procedures to generate detectors for the negative selection algorithm are reviewed and compared in terms of time and space complexity for the production of competent detectors. A new algorithm has been identified and implemented. Experimentation was undertaken, and an analysis is presented on the effectiveness of the various algorithms. The outcome of this empirical analysis reveals that trade-offs have to be made in the choice of algorithm based on the time and space complexities, as well as the detection rate.

1. INTRODUCTION

As engineering and computing problems grow ever more complex, alternative sources of inspiration for solutions to these problems are being sought by computer scientists and engineers. Biology has been seen as a fruitful resource of inspiration with the creation of various biologically inspired techniques such as genetic algorithms, neural networks, and swarm systems (Bentley 2001). The immune system is now receiving more attention and is slowly being realized as a new biologically inspired computational intelligence approach (de Castro and Timmis 2002). An intuitive application of the immune system, and one that many researchers have followed, is to create artificial systems that have the ability to differentiate between self and non-self states: where *self* could be defined as many things, such as, normal behavior, normal network traffic between computers, and so on.

The next section explores one way in which the immune system allows for self non-self discrimination (negative selection), and reviews some approaches in artificial immune systems literature that have attempted to model this process. The main problems with these approaches are highlighted and a new algorithm has been implemented in an attempt to overcome some of these problems. The results presented in this paper demonstrate that the proposed algorithm is equivalent to the exhaustive algorithm for certain classes of problems, and even outperforms it in some cases for example clustered data. The fact still remains that none of the algorithms is able to resolve all the inherent problems associated with detector generation, thus some tradeoffs have to be considered when choosing an algorithm for generating detectors. The final section presents some conclusions and directions for future research.

2. USEFUL IMMUNOLOGY

The immune system is a remarkable and complex natural defense mechanism. The immune system responds to foreign invaders called *pathogens*. The first line of defense is known as innate immunity: this is the immune mechanism our bodies are born with (Janeway 1993). If the innate immune system cannot remove the pathogen, then the adaptive (or acquired) immune system takes over.

The adaptive immune system is made up of B and T-cells, which are capable of responding to certain antigenic patterns presented on the surface of pathogens. Receptors on B and T-cells match antigenic material and depending on the closeness of that match, T-cells stimulate B-cells into rapid proliferation and undergo affinity maturation.

Affinity maturation is a process by which stimulated B-cells are driven to become better tuned to the antigen responsible for initiating the immune response. This enhances the quality of the response (Staines, Brostoff et al. 1994). During affinity maturation, stimulated antibodies undergo a somatic mutation with high rates, termed *hypermutation*. The amount of mutation that a B-cell will undergo is inversely proportional to how well it matches the antigenic pattern: the higher the affinity (match) the lower the mutation, and vice versa. Production of antibodies from these B-cells then ensues, which ultimately remove the antigenic material.

Viewed from a computational perspective, this is an attractive learning mechanism and is one reason why the immune system has attracted such interest.

Pertinent to this work is the maturation of T-cells: what mechanisms are present to prevent the T-cells reacting against the own cells of the body? If this breakdown happens, it is known as an autoimmune disease. This is in part prevented via a process known as negative selection, that allows only the survival of those T-cells that do not recognize self cells. T-cells are produced in the bone marrow, but undergo a maturation process in the thymus gland, after which they are allowed to take part in an immune response. The maturation of the T-cells is conceptually very simple. T-cells are exposed to self-proteins. If this binding activates the T-cell, then the T-cell is killed, otherwise it is allowed into the repertoire. Cells that take part in an immune response are known as immunocompetent cells.

3. ARTIFICIAL IMMUNE SYSTEMS

Artificial immune systems (AIS) are adaptive systems inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving (de Castro and Timmis 2002). The important points of this definition are *inspiration* and *rationale*. In this case, the main idea is to develop problem solving tools that are inspired by the immune system. Through the use of the negative selection process described above, there have been a number of works attempting at building artificial immune systems for virus detection (Forrest, Perelson et al. 1994), computer security (Forrest, Hofmeyr et al. 1996), (Hofmeyr and Forrest 2000) and hardware fault tolerant systems (Bradley and Tyrell 2002). The original work by (Forrest, Perelson et al. 1994), in which the negative selection algorithm was proposed, has been inspirational to almost all the research in the AIS related to the computer security. More recently, that work has also provided the basis for building fault tolerant systems (Tyrell 1999). The basic idea of the algorithm is to produce a set of *change-detectors*, which can detect changes in what is considered normal behavior of a system.

4. NEGATIVE SELECTION: PRINCIPLES AND ISSUES

The negative selection algorithm is inspired by the maturation of T-cells in the thymus gland (Forrest, Perelson et al. 1994). The algorithm consists of two stages: *censoring* and *monitoring*. The censoring phase caters for the generation of change-detectors. Subsequently, the system being protected is monitored for changes using the detectors generated in the censoring stage. However, this algorithm is reported to be very time consuming (D'haeseleer, Forrest et al. 1996), (Wierzchoń 2000). The time taken to generate the detectors is measured by the number of candidate detectors that have to be examined before producing the required number of competent detectors. It was observed that the number of candidate detectors increases exponentially with the size of the self-set, at a fixed probability of not detecting non-self (Forrest, Perelson et al. 1994). This implies that the time to complete the process increases with the size of the self-set. Furthermore, this process does not check for redundant detectors. For minimizing these limitations, some variations of detector generating algorithm were developed: *linear* (D'haeseleer, Forrest et al. 1996), *greedy* (D'haeseleer, Forrest et al. 1996), and *binary template* (Wierzchoń 2000). Both the linear and greedy algorithms run in linear time respective to the size of the self and detector sets (D'haeseleer, Forrest et al. 1996). While the focus of the binary template is to generate efficient non-redundant detectors rather than minimizing the time to generate them. Work in (D'haeseleer, Forrest et al. 1996) claimed that the greedy algorithm manages to resolve this problem by generating a complete repertoire of detectors.

This paper includes the examination of time and space complexities of these algorithms, which were normalized for comparison. In order to cater for worst case situations, all the earlier assumptions included in the derivation of the original time and space complexities were discarded. For a more detailed comparison of several negative selection algorithms, please refer to (Ayara, Timmis et al. 2002).

4.1 EXHAUSTIVE DETECTOR GENERATING ALGORITHM

The exhaustive detector generating algorithm is the original method proposed by (Forrest, Perelson et al. 1994). The algorithm attempts to construct a set of *competent* detectors in the following way: (1) define the self data; (2) generate a random candidate detector; and (3) match each candidate detector generated with self data. If it matches with any self data, it is discarded, otherwise it is added to the collection of competent detectors. A flow diagram of the algorithm is presented in Figure 1 .

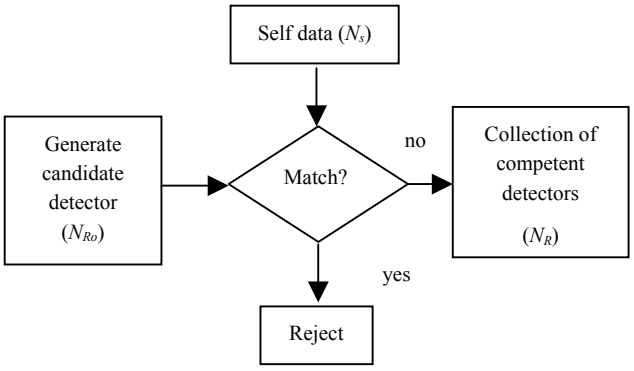

Figure 1: Exhaustive detector generating algorithm.

The time complexity of the algorithm was derived based on two factors: the time to generate a number of candidate detectors (N_{Ro}) and the time to compare each one of them with the population of self-data (Ns). The space complexity depends on the self-population, whose individual members are of length l. In (D'haeseleer, Forrest et al. 1996), the authors derived mathematical formulae to determine the computational complexities of the original algorithm. These were reviewed based on the following considerations: (1) generalising alphabet size m from binary {0,1}, where $m = 2$; and (2) the total number of candidate detectors (N_{Ro}) that can then be generated is m^l, where l is the length of each individual detector string.

Time and space complexities for this algorithm are presented in Section 7, while the empirical experiments carried out with the algorithm using 8-bits binary data, are presented in Section 6. The experiments confirm the limitation observed by (Forrest, Perelson et al. 1994) and (Kim and Bentley 2001), which is a costly computation of generating detectors.

The results motivated the examination and proposal of other algorithms to generate the set of candidate detectors. They are the linear, greedy and binary template algorithms. For the linear and binary template algorithms, please refer to (D'haeseleer, Forrest et al. 1996) and (Wierzchon 2000), respectively. The greedy algorithm will be analyzed in the following section due to its advantages of being linear in relation to the self-set as well as presenting a good coverage of non-self.

4.2 GREEDY DETECTOR GENERATING ALGORITHM

The greedy algorithm improves upon the linear algorithm through the elimination of redundant detectors. Furthermore, it ensures that generated detectors achieve as much coverage of non-self space as possible (D'haeseleer, Forrest et al. 1996). The algorithm is in two phases. The first is the processing phase taken from the linear algorithm, with the second phase being the actual process of generating detectors. This algorithm is based on the use of schemata proposed by (Helman and Forrest 1994) for the r-contiguous bits matching rule. The r-contiguous bits matching rule is a model of the affinity measure in the immune system. Assuming a binary representation of the self and detector strings, the r-contiguous bits matching rule compares a sequence of bits (of length r) in one string with a sequence of bits of the same length in the second string to see if they match. This approach, as shown in Figure 3, has been stated to closely capture the interaction between elements in the immune system (Percus, Percus et al. 1993). This is subject to a pre-defined matching threshold r that is the minimal length of contiguous bits strings common to the two strings for a match to occur. Given this matching rule, the schematic approach is to check for these common sub-strings, as depicted Figure 2, rather than the whole string.

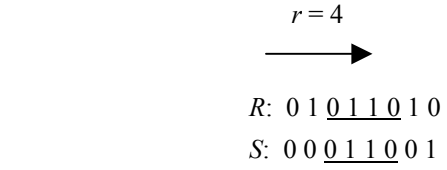

Figure 2: r-contiguous bits matching rule. The strings R and S of length $l = 8$, present $r = 4$ consecutive bits in common.

Assuming a matching threshold r, an alphabet size m (usually binary) and a length l, which is the length of each string, the first phase involves the generation of valid detector *templates* from a total number of m^r possible combinations. Templates are strings with r-contiguous significant bits that start from a specified bit position; and $l - r$ insignificant bits replaced with don't cares. Each template is constructed from a sequence of r bits that can be extended to fully specified detector strings. The set of valid templates are based on the self-data, such that only templates with no match in self are generated. These templates make up the first template array T_S where the nonzero entries constitute the valid templates.

During the second phase, detectors are generated through the extension of the templates to fully specified detector strings. After the generation of each detector string, all the templates that match the detector are removed from the set of valid templates for generating detectors. This prevents the generation of redundant and inefficient detectors at each step.

The time complexity of this algorithm depends on three factors: (1) the time to generate each valid detector templates in m^r; (2) the time to extend each valid template to a fully specified string; and (3) the time to update the templates T_R when creating each detector. The original time and space complexities were derived given these considerations (D'haeseleer, Forrest et al. 1996), but their corresponding mathematical formulae were derived based on the assumptions that each element of the template array can be evaluated in constant time. Also, the analysis ignored the earlier processing phases, before the valid number of detector templates are derived. Additionally, emphasis on binary alphabets can be extended to an alphabet size of m. These were incorporated into the reviewed formulae in Table 5.

5. NEGATIVE SELECTION WITH MUTATION

Work in (de Castro and Timmis 2002) proposed a slight modification of the exhaustive stage of the negative

selection, by introducing somatic hypermutation. Briefly, the procedure proposed the following: (1) define self data; (2) generate a candidate detector randomly; and (3) match each candidate detector with self data, if it matches, perform guided mutation on detector away from self. The guided mutation is performed on the candidate detector, which matches the self data. Mutation is then performed on the parts of the candidate detector that match with the self element. The mutation is adaptive, based on the affinity of the closely matching self element to a candidate detector. This means that the probability of mutation is directly proportional to affinity. Thus, the greater the affinity, the higher is the mutation probability. This idea was taken from the affinity maturation process of B-cells to antigenic patterns in the immune system. In this algorithm, however, the reference is the self-set, instead of non-self set. Hence, the mutation is performed proportionally to affinity to self-set, such that the candidate detector is changed so as not to match self-set. Also, this mutation approach was further augmented by the introduction of a life time indicator for a candidate detector. This in effect restricted the number of times mutation is performed on a candidate detector before a non-improved mutant is discarded. It was thought to have the desired effect of reducing the search space and hence, the number of candidate detectors generated.

The time complexity of NSMutation depends on the following factors: (1) the time to generate a random detector (each of length l) and compare with the population of self data to determine if they match; (2) assuming the use of r-contiguous bits matching, time to mutate matching region of length r in random detector away from self; and (3) a check for redundant detectors. In the worst case, m^l possible detectors can be generated when an alphabet size m is assumed. Hence N_{Ro} candidate detectors are equivalent to m^l. Also, mutation is limited to a region of length r in the candidate detector, which gives the upper bound of mutating the candidate detector as m^r. Subject to these factors, the time and space complexities are given in Table 5.

6. EXPERIMENTS

In order to verify the claims made in (Forrest, Perelson et al. 1994) and (Kim and Bentley 2001) with regards to the exhaustive algorithm, and additionally to test the efficacy of the proposed algorithms, experiments were undertaken using an 8-bit binary data test set. The exhaustive algorithm was used as the empirical standard for the experiments.

6.1 EXPERIMENTAL SETUP

The experiments were performed with randomly generated 8-bits data, with the inclusion of relevant parameters. The following subsections describe the procedures carried out for experimental set up.

6.1.1 Generating self data

As earlier stated, the 8-bit data used were randomly generated. The pseudorandom number generator of the Java 2 Platform (Standard Edition version 1.3) API was used to generate integer numbers between 0 and 255, which were then converted to 8-bit binary strings. During the experiments, there was a need to generate different sizes of self set. This was carried out by creating separate files for different population sizes of self sets.

6.1.2 Setting the matching threshold

The affinity between these binary strings (for the self-set, detector set and test data) was determined using the r-contiguous bits matching rule. The optimal value for matching threshold (r) had to be obtained by changing values of r from 1 to l. This process was done in order to obtain the combined values of correct and incorrect classification by detectors generated using a specific threshold. Correct classification value is derived from the sum of true positive (rate at which non-self is correctly detected) and true negative (rate at which self is correctly not detected). While incorrect classification is the sum of false positive (rate at which self is incorrectly detected) and false negative (rate at which non-self is not detected). Both the correct and incorrect classification values are used to determine the appropriate values of r. This is different from the approach used by (Kim and Bentley 2001) as well as the suggested method in (D'haeseleer, Forrest et al. 1996). In (Kim and Bentley 2001), the value of r was determined from the equations in (Forrest, Perelson et al. 1994), which yielded poor values of matching threshold for the corresponding data. While (D'haeseleer, Forrest et al. 1996) proposed an approach based on the greedy algorithm. Both approaches reveal that there is no hard-and-fast rule for setting this parameter, rather various values can be tested in order to select the optimal one. The following procedure was carried out to determine this parameter:

1. Generate self and test sets from the data sets being experimented upon;
2. Generate required detectors N_R (using equation: $N_R = \frac{-\ln(P_f)}{P_m}$) for different values of r which are varied from 1 to l;
3. Test the detectors generated on the test file to obtain their correct and incorrect classification rates;
4. Use the value of r for which there is minimal incorrect classification and maximum correct classification rates in subsequent experiments.

An outcome of the procedure is illustrated in Table 1 based on the mean values of correct and incorrect classification rates obtained over 10 trials using the following parameters: self set $N_S = 8$, test set $N_T = 256$, available (N_R), and potential (N_{Ro}) repertoires. Given this table, a matching threshold value of 8 will be preferable to the other values since it yields maximum correct and minimum classification rates. When the matching threshold was set to values below 3, no detectors could be generated. This indicates that at such threshold values, all the detectors match the strings in the self-set. The value of *r* thus determines the proper partitioning of the data space into self and non-self segments. This makes the choice of an optimal value for *r* crucial to the effectiveness of the change-detection function.

Table 1: Test for obtaining optimal value of matching threshold (*r*)

r	N_{Ro}	N_R	Correct classification rates	Incorrect classification rates
3	209.2	5	41.80%	58.20%
4	37.30	12	56.02%	43.98%
5	46.50	29	70.98%	29.02%
6	87.90	74	85.12%	14.88%
7	210.90	196	89.10%	10.90%
8	604.80	589	91.72%	8.28%

6.1.3 Mutation probability

The mutation probability *(mutProb)* is a threshold that determines whether a bit position in a binary string will be mutated or not. This value was initially implemented using an adaptive mechanism which is calculated as the length of the matching bits in two binary strings divided by the length *l* of the binary string. The value generated is a real number between 0.0 and 1.0. This threshold value is then used to determine whether a bit position is subjected to mutation. For each bit position to be mutated, if a randomly generated number between 0.0 and 1.0 is less than the mutation probability, the bit is mutated. The converse is the case when the random number is greater than the mutation probability. In (Ayara, Timmis et al. 2002), the adaptive mutation probability was discovered to degrade the time complexity of the algorithm if the probability is greater than a specific value. This is because the probability indicates that a sizeable fraction of the total number of bits in a random binary string matches self. Hence the process of mutating a random detector is restricted to limited options. This can be explained by a matching threshold $r = 8$. In this case, the mutation probability is 1 and the process of mutation just flips a random detector to its image. In a situation that the image also matches self, mutation flips back to the original detector which also matches self. If this is the case for a significant number of random detectors generated, the time complexity is increased considerably. However there is a threshold value below which this will not occur. For example, the results of experiments in (Ayara, Timmis et al. 2002) show that using 8-bits binary data generated randomly the maximum mutation probability that will not make the algorithm worse off than the original exhaustive, for threshold values of 7 and 8, was confirmed to be 0.8. This directed the choice of mutation probability for subsequent experiments, which was set to 0.5.

6.1.4 Detector life-time indicator

The detector life-time indicator (*mutLim*) determines the number of attempts that mutation can be performed on a random detector. When values of this parameter are greater than 1, it was found to increase the time complexity of NSMutation when used with adaptive mutation probability. This phenomenon can be linked to the explanation given in section 6.1.3, which accounts for the poor behaviour of the algorithm using adaptive mutation probability. In a situation when the mutation probability is above a specific value, and the limited detector options that mutation can generate also match with self, an increase in life-time indicator only extends the time for the flipping the detector back and forth between the image and the original detector.

Some definitions of terms used in the experiments are listed in Table 2.

Table 2: Definitions of terms used in experiments

Terms	Definitions
l	Length of string
r	Matching threshold
m	Alphabet size
N_S	Population of self data
N_{Ro}	Population of candidate detectors
N_R	Population of competent detectors
P_m	Probability of detecting a non-self
P_f	Probability of failing to detect non-self
N_T	Population of test data
mutProb	Mutation probability
mutLim	Mutation limit (Detector life-time indicator)

6.2 THE BOTTLENECK FOR NEGATIVE SELECTION

Given the earlier discussion regarding the constraint of the exhaustive algorithm, i.e., the size of the set of candidate detectors increases exponentially with the size of the self-set, initial tests were performed to check if this claim holds true for the proposed algorithm. This process involved determining the number of candidate detectors required to produce a specified number of competent detectors when the population size of self is increased progressively. The test was carried out with both the NSMutation and exhaustive algorithm for comparison.

Using the definitions provided in Table 2, the mathematical equations for estimating P_m, N_R, and N_{Ro} (Forrest, Perelson et al. 1994), were employed for implementing the algorithm.

The following procedures were carried out for NSMutation algorithm:

1. For a particular data set, derive r (section 6.1.2) for all runs of the experiment;
2. Calculate P_m and select a desired value for P_f ;
3. Determine the value of N_R according to the following equation: $N_R = \frac{-\ln(P_f)}{P_m}$;
4. Set the values of *mutProb* and *mutLim* using the guidelines in sections 6.1.3 and 6.1.4 respectively;
5. Execute steps a-c a number of times while incrementing the size of N_S, $8 \le N_S \le 160$. (The selected value was 100 for trial runs):
    a. Determine N_{Ro} experimentally by generating random strings until N_R valid detectors are determined;
    b. Once a match occurs between a self string and a candidate detector, or there is a duplicate of the detector in the detector set, perform uniform mutation in a guided manner until the candidate detector becomes a competent detector. The detector is then added to the set of useful detectors;
    c. The number of mutation attempts is limited by a detector life time indicator (*mutLim*), which is set to a fixed value.

This life-time indicator constrains the time expended to change a detector that closely resembles self. In a situation where a mutated detector is not improved by the time the life-time has expired, it is discarded and replaced by another random detector. The same process was undertaken for the exhaustive algorithm, excluding the mutation operator and the check for redundant detector in the detector set. The *potential repertoire size* (N_{Ro} - collection of candidate detectors before negative selection) for both algorithms was recorded for comparison. While the population of detectors generated after negative selection, known as the *available repertoire size* (N_R), was set as a parameter for the simulation. The results obtained from the experiments are presented in Table 3 and Figure 3. These results are obtained from 100 trials for each size of the self-set, $8 \le N_S \le 160$, with the following parameters $N_R = 589$, $r = 8$, $P_f = 0.1$, *mutLim* = 4, *mutProb* = 0.5. Each column of Table 3 holds values calculated as a mean of the number of trials, while the standard deviations are enclosed in brackets for each mean value. Column (a) indicates the size of self set, (b) holds the theoretical estimates of potential repertoire (N_{Ro}), (c) the experimental N_{Ro} values for the exhaustive algorithm, (d) experimental N_{Ro} values for NSMutation, and (e) the mean mutation occurrence over 100 trials.

The results in Table 3 are selected from the outcome of the experiments shown in Figure 3. From Table 3, it can be clearly seen that the potential repertoire generated for both algorithms are similar, for example when the population size of self set is 152, the exhaustive and NSMutation algorithms generate potential repertoire of 1128.16 and 1127.62 respectively. This explains the overlap in the graphs of both algorithms. Also, column (e) in Table 3 show that mutation occurs 1.807 ≈ 2 times out of 100 trials.

In order to determine the effectiveness of the NSMutation in comparison to the exhaustive algorithm, their detection rates were tested empirically using a single population size of self $N_S = 8$. Other parameter values include $N_R = 589$, $r = 8$, $P_f = 0.1$, *mutLim* = 4, *mutProb* = 0.5, $N_T = 256$. The outcomes of these tests are presented in Table 4.

As shown in Table 4, the theoretical estimation of potential repertoire size is calculated as 608.21, while the mean potential repertoire sizes for exhaustive and NSMutation respectively are 608.10 and 608.40. Their corresponding detection rates are 90.36% for exhaustive and 89.84% for NSMutation. Testing the statistical difference between their detection rates using the Z-test, gave a value of +0.085, which shows that their detection rates are not statistically different.

Table 3: Experimental results generated from 8-bits data based on 100 trials for self set N_S = 152, 160.

N_S	N_{Ro} (Theoretical)	N_{Ro} (Exhaustive algorithm)	N_{Ro} (NSMutation algorithm)	Mutation Occurrence
(a)	(b)	(c)	(d)	(e)
152	1068.62	1128.16 (33.130)	1127.62 (31.739)	1.807 (1.020)
160	1102.61	1084.26 (32.344)	1091.14 (31.190)	1.768(0.992)

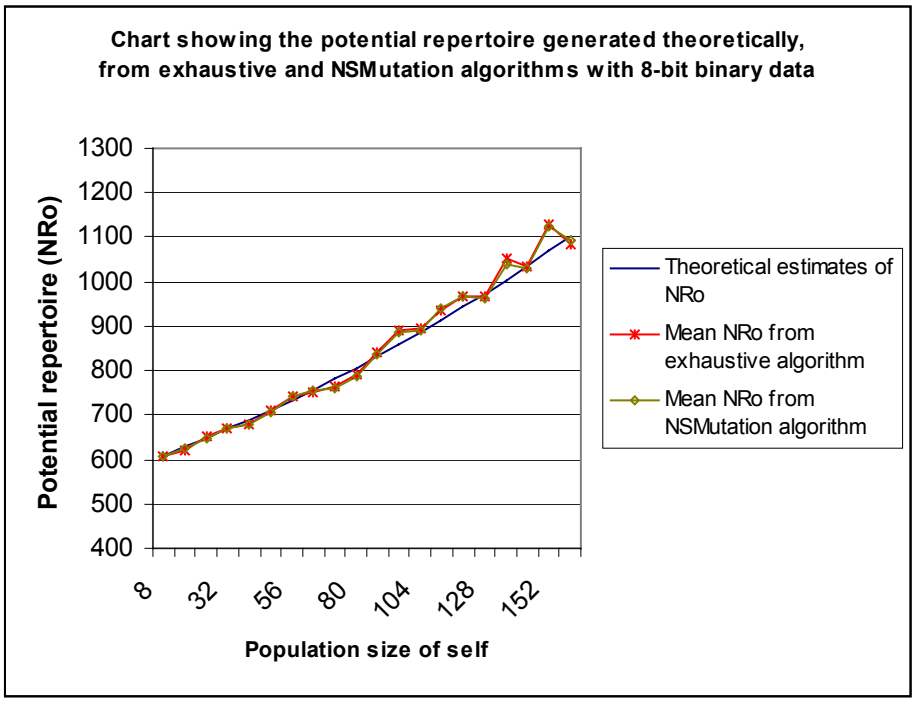

Figure 3: Chart for 8-bits data that illustrates the theoretical estimates and mean population of potential repertoire based on 100 trial runs for both algorithms given each size of self-set.

Table 4: Test results of detection performance of the NSMutation and exhaustive algorithms over 10 trials.

	Exhaustive	NSMutation
Theoretical N_{Ro}	608.21	
Experimental N_{Ro}	608.10	608.40
Detection rates	90.36%	89.84%
Z-test value	+0.085	

7. ANALYSIS AND DISCUSSION

In this section, the output of the experiments performed in section 6.2 are analyzed with the aim of discussing the features peculiar to NSMutation and comparing with the exhaustive algorithm in terms of tests conducted. This comparison is then extended to the other algorithms for an overview of all the detector generating algorithms.

From the diagram of Figure 3, it can be observed that the number of candidate detectors examined for the exhaustive algorithm increases exponentially with the size of the self-set. This confirms the limitation expressed by (Kim and Bentley 2001). This behavior is also exhibited by NSMutation, whose pattern of increase in potential repertoire closely resembles that of the exhaustive algorithm. This can be explained to be a result of the random nature of the self set, which is normally distributed. During the process of mutating a candidate detector for the NSMutation algorithm, the aim is to guide the candidate detector away from self set. But since the self set is randomly distributed around the search space, there is an equal probability of mutating the random detector away from or towards self set. Hence the impact of guided mutation cannot be guaranteed for random data, and the outcome is more or less a random generation of detectors. However, this is not the usual case for a clustered self set with well-defined boundaries. Preliminary experiments performed in (Ayara, Timmis et al. 2002) to test this showed that the potential repertoire is almost linear with increase in the self set. Also refer to (Ayara, Timmis et al. 2002) for the pseudocodes of all the algorithms.

The comparison of their detection rates in Table 4 further confirms the similarities. The difference in their performance at detecting non-self was evaluated using the Z-statistic at a significance level of 0.05%, and the outcome showed that their detection rate performances were not statistically different.

Although from Figure 3 it can be asserted that the NSMutation algorithm behaves similarly to the exhaustive, some extensive studies of the NSMutation algorithm (Ayara, Timmis et al. 2002), provide more information about some parameters of the algorithm that control its performance. They include the matching threshold (*r),* detector life-time rate (*mutLim*) and mutation probability (*mutProb*). These parameters can deteriorate its performance than its predecessor or speculatively better, if a good combination of parameters for the data set can be obtained. For example, when $r = l$ (length of each string), the effect of the *mutProb* on the time complexity is more profound, even though there is a higher chance of generating good detectors due to the exact matching. The effect of *mutProb* is aggravated by a high value of *mutLim*. For example when $l = 8$, $r = 8$, *mutLim* = 4, and *mutProb* = 1.0, mutation of a non-competent detector produces its image and if the mutant also matches self, further mutation just flips the image back to the original detector, thereby causing an alternation between the image and the original detector. The *mutLim* parameter thus causes this process to be carried out for a specific number of trials. However, as $r << l$, the effect of *mutProb* and *mutLim* pale into insignificance, since the value of r already triggers high time complexity. This parameterization factor for good performance of learning algorithms has been observed by (Bentley, Gordon T. et al. 2001). So the next question to be answered is "what parameter values for the NSMutation algorithm can make it outperform the exhaustive?"

Altogether, the reviewed and normalized time and space complexities of all the algorithms, as shown in Table 5, reveal the characteristics in terms of computational complexity. While the time complexities of the exhaustive algorithm and NSMutation are exponential with respect to the size of self, the others have time complexities that are linear functions of the self. The linear algorithm, however, has the disadvantage of generating redundant detectors, as is the case with the exhaustive; this in turn limits its performance. However, the greedy algorithm achieves the best coverage for detection, due to the fact that it generates complete repertoires of detectors as claimed by (D'haeseleer, Forrest et al. 1996). The binary template, which derives its inspiration from the greedy also achieves similar coverage. Both greedy and binary template algorithms have higher computational complexity when compared to the linear algorithm. The greedy algorithm includes the process of checking that each detector generated represents a cluster of non-self to prevent redundancy and also ensure that efficient detectors are produced. Also the binary template algorithm includes similar processes of removing redundant detectors and ensuring that inefficient detectors are eliminated. Hence, these additional processes of guaranteeing non-self coverage and non-redundancy incur extra time to complete the algorithms. It must be noted that when the matching threshold r approaches length l of each string in the search space, the linear time complexities of the linear, greedy and binary template with respect to the size of the self-set, may exhibit similar behavior as that of the exhaustive and NSMutation, due to the exponential value m^r in their time complexity equation.

In terms of space complexity, NSMutation has a higher space complexity that the exhaustive. The reason for this is that the NSMutation stores the detectors as they are generated for comparison with subsequent detectors in order to prevent redundancy. On the other hand, the linear, greedy and binary template incur more space complexity due to the storage of m^r binary template strings that are stored and updated. However the binary template algorithm has a lower space complexity when compared with the linear and greedy algorithms.

Another criterion for comparing the algorithms is the coverage of detectors. This factor measures the extent to which the detectors generated from the negative selection algorithm are fully representative of the non-self set. Thus it thereby provides a means of determining the efficiency of the algorithm. If complete coverage is to be achieved, it implies that all non-self detectors must be generated. However, there is a need to maintain a balance between the time taken to generate detectors and getting a good coverage. This balance seems to be best achieved by the greedy

algorithm. The algorithm is able to generate non-redundant detectors that have high detection coverage, at minimal time complexity.

In summary, it can be argued that the NSMutation is more or else the exhaustive algorithm since they expend similar time complexity and achieve as much coverage of non-self. However, NSMutation differs from the exhaustive algorithm because it includes checks for redundancy and tunable parameters that can induce different performance. When compared with the linear, greedy and binary templates, the simplicity of NSMutation makes it quite attractive as against the others that entail cumbersome procedures. Furthermore, only the exhaustive and NSMutation can be used with other matching rules. The linear, greedy and binary template algorithms are restrictive. They are limited to the r-contiguous bits matching rule, which renders them inextensible and inappropriate for other matching rules. The benefits of NSMutation thus include simplicity, high detection rate performance and extensibility.

Table 5: Reviewed time and space complexities of all detector generating algorithms (refer to original equations in (D'haeseleer, Forrest et al. 1996)).

Algorithm	Time	Space
Exhaustive	$O(m^l.N_S)$	$O(l.N_S)$
Linear	$O((l-r+1).N_s.m^r)+$ $O((l-r+1).m^r)+$ $O(l.\ N_R)$	$O((l-r+1)^2.m^r)$
Greedy	$O((l-r+1).N_s.m^r)+$ $O((l-r+1).m^r.N_R)$	$O((l-r+1)^2.m^r)$
Binary Template	$O(m^r.N_S)+$ $O((l-r+1).m^r.N_R)$	$O((l-r+1).m^r)+$ $O(N_R)$
NSMutation	$O(m^l.N_S)+O(N_R.m^r)$ $+O(N_R)$	$O(l.(N_S + N_R))$

8. CONCLUSIONS

This paper has made a comparison between the different negative selection algorithms for generating detectors, and implemented a variation of the initial exhaustive algorithm. The results were presented using the time taken to generate detectors, as well as the detection rate coverage of the final detectors generated. It has been demonstrated that there are trade-offs to be made in deciding on the best algorithm for producing the detectors. The exhaustive algorithm takes considerable time (exponential in size of self data) and produces redundant detectors; the linear algorithm has a linear time complexity but also produces redundant detectors; the greedy algorithm produces a complete repertoire using up as much space as the linear algorithm, but has a higher computational complexity; the binary template produces a minimal set of efficient detectors at the expense of more time complexity; and finally NSMutation is similar to the exhaustive algorithm with the difference of eliminating redundancy and possessing parameters that can be optimized for better performance. However for structured data sets, the NSMutation has shown better performance in terms of time complexity, but there is still need for further verification. Thus, in a case where choice has to be made between both exhaustive and NSMutation, the latter has the advantages of possessing tunable parameters, eliminating redundant detectors, and being suitable for any matching rule. But, the decision lies with the constraints being met while implementing the algorithm in its target domain. Different domains place emphasis on different constraints that must be satisfied. These might include factors such as time to generate detectors; space storage used by the detectors; matching function; as well as the performance of detectors generated. Since no algorithm has managed to minimize all these constraints, trade-offs have to be made in choosing an algorithm for generating negative selection detectors. But it must be said that more analysis of the NSMutation algorithm will need to be carried out in order to determine the best combination of parameters that can improve it significantly.

Acknowledgments

We would like to thank NCR FSG for their continued financial support and valuable input into this research. Leandro N. de Castro would like to thank CNPq (Profix 540396/01-0) for their financial support.

We gratefully acknowledge comments received from anonymous reviewers.

References

M. Ayara, J. Timmis, et al. (2002). An Investigation into Negative Selection for Change Detector Generation, *Technical Report*, Computing Laboratory, University of Kent at Canterbury, U.K.

D. W. Bradley and A. M. Tyrrell (2002). A Hardware Immune System for Benchmark State Machine Error Detection. *Congress on Evolutionary Computation. Part of the World Congress on Computational Intelligence*, 813-818, Honolulu, HI. USA, IEEE,.

P.J. Bentley. (2001) *Digital Biology*. Hodder Headline.

P. J. Bentley, T. Gordon, J. Kim and S. Kumar (2001). New Trends in Evolutionary Computation. T*he Congress on Evolutionary Computation (CEC-2001)*, 162-169, Seoul, Korea, May 27-30, 2001.

L. N. de Castro and J. I. Timmis (2002). Artificial Immune Systems: A New Computational Intelligence Approach, Springer-Verlag.

P. D'haeseleer (1996). An Immunological Approach to Change Detection: Theoretical Results. *IEEE Computer Security Foundations Workshop*, Dromquinna Manor, County Kerry, Ireland, June 10-12, 1996.

P. D'haeseleer, S. Forrest, et al. (1996). An Immunological Approach to Change Detection. In *Proc. of IEEE Symposium on Research in Security and Privacy*, Oakland, CA.

S. Forrest, S. A. Hofmeyr, et al. (1997). Computer Immunology, *Communications of the ACM*, 40(10):88-96.

S. Forrest, A. Perelson, et al. (1994). Self Nonself Discrimination in a Computer. In *Proc. of IEEE Symposium on Research in Security and Privacy*, 202-212, Oakland, CA, May 16-18, 1994.

P. Helman and S. Forrest(1994). An Efficient Algorithm for Generating Random Antibody Strings. *Technical Report CS-94-07*, The University of New Mexico, Albuquerque, NM.

S. Hofmeyr and S. Forrest (2000). Architecture for an Artificial Immune System. *Evolutionary Computation*, 7(1):45-68.

C. A. Janeway (1993). How the Immune System Recognizes Invaders. *Scientific American*: 41-47.

J. Kim and P. Bentley (2001). Investigating the roles of Negative Selection and Clonal Selection in an Artificial Immune System for Network Intrusion Detection, *Technical Report*, Dept. of Computer Science, UCL, London.

J. K. Percus, O. E. Percus, et al. (1993). Predicting the Size of T-cell Receptor and Antibody Combining Region from Consideration of Efficient Self-Nonself Discrimination. *National Academy of Science*, 90:1691-1695.

N. Staines, J. Brostoff, et al. (1994). Introducing Immunology, Mosby.

A. M. Tyrrell (1999). Computer Know Thy Self!: A Biological Way to Look at Fault-Tolerance. *Second Euromicro/IEEE Workshop on Dependable Computing Systems*, 129-135, Milan, Italy.

S. T. Wierzchon(2000a). Generating Optimal Repertoire of Antibody Strings in an Artificial Immune System. In M. Klopotek, M. Michalewicz and S. T. Wierzchon (eds.) *Intelligent Information Systems*. Advances in Soft Computing Series of Physica-Verlag/Springer Verlag, Heidelberg/New York 2000, Physica-Verlag, 119-133.