An Immunity-Based Technique to Characterize Intrusions in Computer Networks

Dipankar Dasgupta, *Associate Member, IEEE*, and Fabio González, *Student Member, IEEE*

***Abstract*—This paper presents a technique inspired by the negative selection mechanism of the immune system that can detect foreign patterns in the complement (nonself) space. In particular, the novel pattern detectors (in the complement space) are evolved using a genetic search, which could differentiate varying degrees of abnormality in network traffic. The paper demonstrates the usefulness of such a technique to detect a wide variety of intrusive activities on networked computers. We also used a positive characterization method based on a nearest-neighbor classification. Experiments are performed using intrusion detection data sets and tested for validation. Some results are reported along with analysis and concluding remarks.**

***Index Terms*—Artificial immune system, biological systems modeling, detector generation, genetic algorithms, intrusion detection.**

I. Introduction

THE NATURAL immune system is a subject of great research interest because of its information processing capabilities [9], [19]. In particular, it performs many complex computations in a parallel and distributed fashion. Like the nervous system, the immune system can learn new information, recall previously learned information, and perform pattern recognition tasks in a decentralized fashion. Rather than relying on a central controller, the immune system, however, uses a distributed detection and response mechanism in order to respond to foreign invaders.

The security in computing may be considered as analogous to the immunity in natural systems. In computing, threats and dangers (of compromising privacy, integrity, and availability) may arise because of a malfunction of components or intrusive activities (both internal and external). The idea of using immunological principles in computer security [8], [10], [19], [20] has progressed since 1994. Forrest and her group at the University of New Mexico have worked toward a long-term goal of building an artificial immune system for computers. In their approach, the problem of protecting computer systems from harmful viruses was viewed as an instance of the more general problem of distinguishing self (legitimate users, uncorrupted data, etc.) from dangerous other (unauthorized users, viruses, and other malicious agents). This method (called the negative-selection algorithm) was used to detect changes in the protected data and program files. In another work, they applied the algorithm to monitor UNIX processes, where the purpose was to detect harmful intrusions in a computer system. Kephart [20] suggested another immunologically inspired approach (decoy program) for virus detection. In this approach, known viruses were detected by their computer-code sequences (signatures) and unknown viruses by their unusual behavior within the computer system. Most of these methods tried to address security issues with immunological approaches.

The problem of characterizing the normal and abnormal behavior of a system in network environment is very complex. The general assumption is that the normal behavior of a system can often be characterized by a series of observations over time. Also, normal system behavior generally exhibits stable patterns when observed over a period of time. There are multiple approaches to such anomaly detection [5], [7], [13]–[15], [22], [23], [26] and most of them work by building a model or profile of the system that reflects its normal behavior. A simple approach is to define thresholds (upper and lower) for each monitored parameter of the system and, if a parameter exceeds this range, it is considered an abnormality. The most common approach uses a statistical model [13], [15] to calculate the probability of occurrence of a given value; the lower the probability, the higher the possibility of an anomaly. In general, statistical approaches model individually different variables that represent the state of the system.[1] This approach, however, ignores two important facts.

1) *Normalcy depends on time:* A value that might be considered normal at a given time might be abnormal at a different time. In general, we must discuss normal (or abnormal) temporal patterns instead of normal (or abnormal) values [23].
2) *The notion of normalcy depends on correlations among different parameters:* The independent values of two different parameters might be considered normal, but their combination might show abnormality or otherwise [25].

Other approaches also build models to predict the future behavior of systems or processes based on the present and past states [4], [5], [14], [17], [23]. Accordingly, if the actual state of the system differs considerably from the predicted state, an anomaly alarm is raised. These approaches are more successful in capturing temporal and multiple-variable correlations. However, more time is needed for training the model and, in some

Manuscript received January 5, 2001; revised July 20, 2001 and February 9, 2002. This work was supported in part by the Defense Advanced Research Projects Agency under Contract F30602-00-2-0514 and the National Science Foundation under Grant IIS-010425.

D. Dasgupta is with the Computer Science Division, Mathematical Sciences Department, the University of Memphis, Memphis, TN 38152 USA (e-mail: ddasgupt@memphis.edu).

F. González is with Computer Science Division, Mathematical Sciences Department, the University of Memphis, Memphis, TN 38152 USA (e-mail: fgonzalz@memphis.edu) and also with Departamento de Ingeniería de Sistemas, Universidad Nacional de Colombia, Ciudad Universitaria, Bogotá, Colombia.

Publisher Item Identifier S 1089-778X(02)06066-6.

[1] Despite the posibility for using multivariate distributions, the assumptions are too restrictive to be applied to real problems.

1089-778X/02$17.00 © 2002 IEEE

cases, its application can be infeasible because of the size of data sets involved (generally, this is the case with network security).

Therefore, the challenge is to build an anomaly detection system that can capture multivariable correlations and is capable of dealing with the large amount of data generated in a computer network environment. Data-mining techniques have been applied with some success to this problem [22], [24], [25]. This approach has the advantages of dealing with large data sets and being able to garner useful knowledge (generally expressed in terms of rules). For these techniques, it is important that the data have some degree of structure. In several works, the network traffic data (packet level) is processed to get connection information (type, duration, number of bytes transmitted, etc). In some cases, information about which connections are normal and which connections are attacks is necessary for some algorithms.

This paper proposes an approach that does not rely on structured representation of the data and uses only positive data to build a normal profile of the system. It is applied to perform anomaly detection for network security, but it is a general approach that can be applied to different anomaly detection problems.

First, a positive characterization (PC) technique is presented. It is applied to different data sets and the results are analyzed. Later, a negative characterization (NC) technique is proposed that alleviates the efficiency issue of the PC technique. This technique is inspired by artificial immune systems ideas and it attempts to extend Forrest's (self–nonself) [14], [16] two-class approach to a multiclass approach. Specifically, the nonself space will be further classified in multiple subclasses to determine the level of abnormality. Experiments are performed and the results are compared with the ones produced by the PC technique.

A. Anomaly Detection Problem Definition

The purpose of anomaly detection is to identify which states of a system are normal and which are abnormal. The states of a system can be represented by a set of features.

Definition 1—System State Space: A state of the system is represented by a vector of features $x^i = (x_1^i, \ldots, x_n^i) \in [0.0, 1.0]^n$. The space of states is represented by the set $S \subseteq [0.0, 1.0]^n$. It includes the feature vectors corresponding to all possible states of the system.

The features can represent current and past values of system variables. The actual values of the variables could be scaled or normalized to fit a defined range [0.0, 1.0].

Definition 2—Normal Subspace (Crisp Characterization): A set of feature vectors $\mathrm{Self} \subseteq S$ represents the normal states of the system. Its complement is called $\mathrm{Nonself}$ and is defined as $\mathrm{Nonself} = S - \mathrm{Self}$. In many cases, we will define the $\mathrm{Nonself}$ (or $\mathrm{Nonself}$) set using its characteristic function $\chi_{\mathrm{self}} : [0.0, 1.0]^n \rightarrow \{0, 1\}$

$$\chi_{\mathrm{self}}(\vec{x}) = \begin{cases} 1, & \text{if } \vec{x} \in \mathrm{Self} \\ 0, & \text{if } \vec{x} \in \mathrm{Nonself} \end{cases}.$$

The terms *self* and *nonself* are motivated by the natural immune system. In general, there is no sharp distinction between the normal and abnormal states; instead, there is a degree of normalcy (or, conversely, abnormality). The following definition reflects this:

Definition 3—Normal Subspace (Noncrisp Characterization): The characteristic function of the normal (or abnormal) subspace is extended to take any value within the interval $[0.0, 1.0] : \mu_{\mathrm{self}} : [0.0, 1.0]^n \rightarrow [0.0, 1.0]$. In this case, the value represents the degree of normalcy: "1" indicates normal, "0" indicates abnormal, and the intermediate values represent elements with some degree of abnormality.[2]

The noncrisp characterization allows a more flexible distinction between normalcy and abnormality. However, in a real system, it may be necessary to decide when to raise an alarm or not. In this case, the problem becomes again a binary decision problem. It is easy to go from the noncrisp characterization to the crisp one by establishing a threshold

$$\mu_{\mathrm{self},t}(\vec{x}) = \begin{cases} 1, & \text{if } \mu_{\mathrm{self}}(\vec{x}) > t \\ 0, & \text{if } \mu_{\mathrm{self}}(\vec{x}) \leq t \end{cases}.$$

Definition 4—Anomaly Detection Problem: Given a set of normal samples $\mathrm{Self}' \subseteq \mathrm{Self}$, build a good estimate of the normal space characteristic function χ_{self} (or μ_{self} in the noncrisp case). This function should be able to decide whether or not the observed state of the system is anomalous.

In the following, we will describe a PC approach (using the nearest neighbor distance) and a negative approach, respectively.

II. PC Approach

In this approach, we used the positive samples to build a characterization of the Self space. In particular, we did not assume a model for the Self set. Instead, we used the positive sample set itself[3] for a representation of the Self space. The degree of abnormality of an element is calculated as the distance from itself to the nearest neighbor in the Self set. We chose to define the characteristic function of the $\mathrm{Nonself}$ set, since its definition is more natural, and the derivation of the Self set characteristic function is straightforward

$$\mu_{\mathrm{nonself}}(\vec{x}) = D(\vec{x}, \mathrm{Self}) = \min\{d(\vec{x}, \vec{s}) : \vec{s} \in \mathrm{Self}\}.$$

Here, $d(x, s)$ is a Euclidean distance metric (or any Minkowski metric[4]). $D(\vec{x}, \mathrm{Self})$ is the nearest neighbor distance, i.e., the distance from x to the closest point in Self. Then, the closer an element $\vec{x}$ is to the self set, the closer the value of $\mu_{\mathrm{nonself}}(\vec{x})$ is to zero.

The crisp version of the characteristic function is the following:

$$\begin{aligned}\mu_{\mathrm{nonself},t}(\vec{x}) &= \begin{cases} 1, & \text{if } \mu_{\mathrm{self}}(\vec{x}) > t \\ 0, & \text{if } \mu_{\mathrm{self}}(\vec{x}) \leq t \end{cases} \\ &= \begin{cases} 1, & \text{if } D(\vec{x}, \mathrm{Self})) > t \\ 0, & \text{if } D(\vec{x}, \mathrm{Self})) \leq t \end{cases}.\end{aligned}$$

[2] This definition is basically a fuzzy-set specification. In fact, the function μ_{self} is a membership function. However, we chose not to refer to it directly, because of the different emphasis of this paper.

[3] This approach is known as *lazy learning* and it is used commonly in classification algorithms.

[4] In our experiments, we also used the D_∞ metric defined by $D_\infty(\vec{x}, \vec{y}) = \max(|x_1 - y_1|, \ldots, |x_n - y_n|)$.

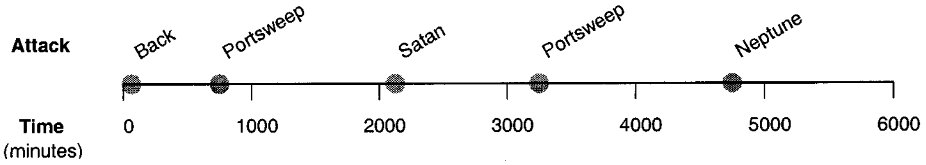


Fig. 1. Network attacks on the second weekend.

In a dynamic environment, the parameter values that characterize normal system behavior may vary within a certain range over a period of time. The term $(1-t)$ represents the amount of allowable variability in the self space (the maximum distance that a point can be from the Self samples to be considered as normal).

This PC can be implemented efficiently using spatial trees. In our implementation, a KD-tree [2], [3], [18] was used. A KD-tree represents a set of k-dimensional points and it is a generalization of the standard one-dimensional binary search tree. The nodes of a KD-tree are divided into two classes: 1) internal nodes partition the space with a cut plane defined by a value in one of the k dimensions 2) and external nodes (leaves) define "buckets" (resulting in hyperrectangles), where the points are stored.

This representation allows answering queries in an efficient way. The amortized cost of a nearest neighbor query is $O(\log N)$ [3]. We used a library (that implements the KD-tree structure) developed at the University of Maryland [27].

A. PC Experiments

We performed experiments with real intrusion data obtained from the Lincoln Laboratory of the Massachusetts Institute of Technology[21]. These data represent both normal and abnormal information collected in a test network, where simulated attacks were performed. The purpose of these data is to test the performance of intrusion detection systems. The data sets (corresponding to the year 1999) contain complete weeks with normal data (not mixed with attacks). This provides enough samples to train our detection system.

The test data set is composed of network traffic data (tcpdump, inside and outside network traffic), audit data (bsm), and file systems data. For our initial set of experiments, we used only the outside tcpdump network data for a specific computer (e.g., hostname: marx) and then we applied the tool *tcpstat* to get traffic statistics. We used the first week's data for training (attack free) and the second week's data for testing, which include some attacks. Some of these were network attacks, the others were inside attacks. Only the network attacks were considered for our testing. These attacks are described in Table I and the attack timeline is shown in Fig. 1.

Three parameters were selected to detect some specific type of attacks. These parameters were sampled each minute (using *tcpstat*) and normalized. Table II lists six time series S_i and T_i for training and testing, respectively.

The set S of normal descriptors is generated from a time series $R = \{r_1, r_2, \ldots, r_n\}$ in an overlapping sliding window fashion

$$S = \{(r_1, \ldots, r_w), (r_2, \ldots, r_{w+1}), \ldots, (r_{n-w+1}, \ldots, r_n)\}$$

where w is the window size. In general, from a time series with n points, a set of $n-w+1$ of w-dimensional descriptors can be generated. In some cases, we used more than one time series to generate the feature vectors. In those cases, the descriptors were put side by side in order to produce the final feature vector. For instance, if we used the three time series S1, S2, and S3 with a window size 3, a set of nine-dimensional feature vectors was generated.

In each experiment, the training set was used to build a KD-tree to represent the self set. Then, the distance (nearest neighbor distance) from each point in the testing set to the self set was measured to determine deviations.

For this set of experiments, the variables were considered independently, i.e., the feature vectors were built using only one variable (time series) each time. Fig. 2 shows an example of the training and testing data sets for the parameter *number of packets per second*. Fig. 3(a) represents the Nonself characteristic function $\mu_{\text{nonself}}(\vec{x})$, i.e., the distance from the test set to the training set for the same parameter. In this case, the window size used to build the descriptors was 1. Figs. 3(b) and (c) show $\mu_{\text{nonself}}(\vec{x})$ for using a window size of 3. In Fig. 3(b), the Euclidean distance is used and in Fig. 3(c) the D_∞ distance is used.

The plots of the nonself characteristic function show some peaks that correspond to significant deviations from the normal. It is easy to check that these peaks coincide with the network attacks present on the testing data (see Table I and Fig. 1). We conclude the following from the results.

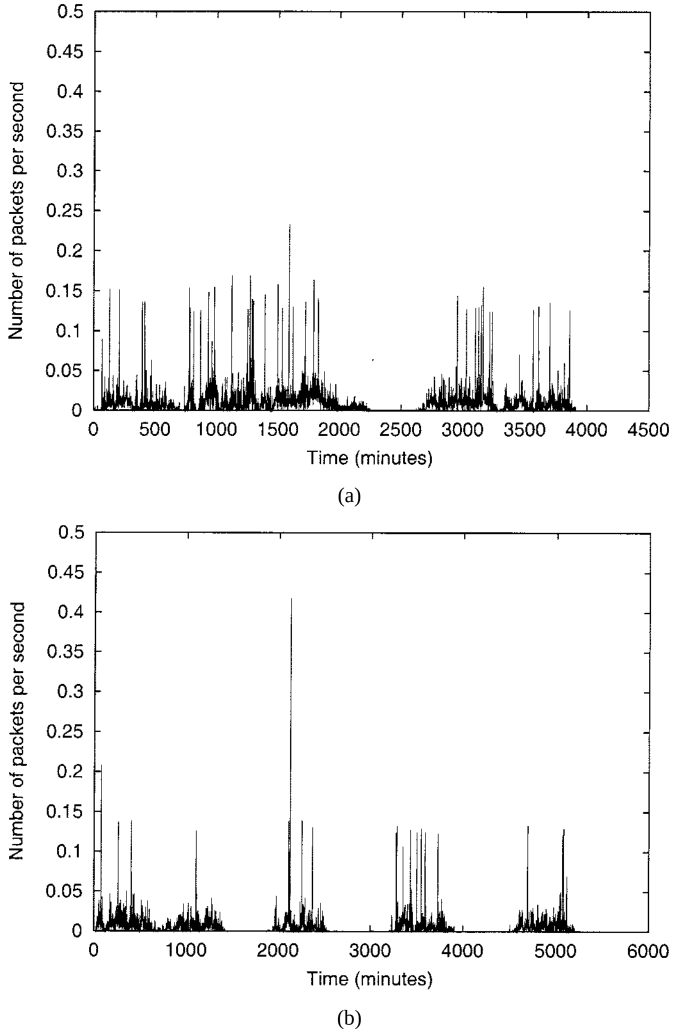
1) Using only one parameter is not enough to detect the five attacks. Fig. 3 shows how the function $\mu_{\text{nonself}}(\vec{x})$ detects deviations that correspond to attacks. However, none of the parameters is able to detect, independently, all five attacks.

TABLE I
SECOND WEEK ATTACKS DESCRIPTION

Day	Attack Name	Attack Type	Start	Duration
1	Back	DOS	9:39:16	00:59
2	Portsweep	PROBE	8:44:17	26:56
3	Satan	PROBE	12:02:13	02:29
4	Portsweep	PROBE	10:50:11	17:29
5	Neptune	DOS	11:20:15	04:00

TABLE II
DATA SETS AND PARAMETERS USED

Name	Description	Week	Type
S1	Number of bytes per second	1	Training
S2	Number of packets per second	1	Training
S3	Number of ICMP packets per second	1	Training
T1	Number of bytes per second	2	Testing
T2	Number of packets per second	2	Testing
T3	Number of ICMP packets per second	2	Testing



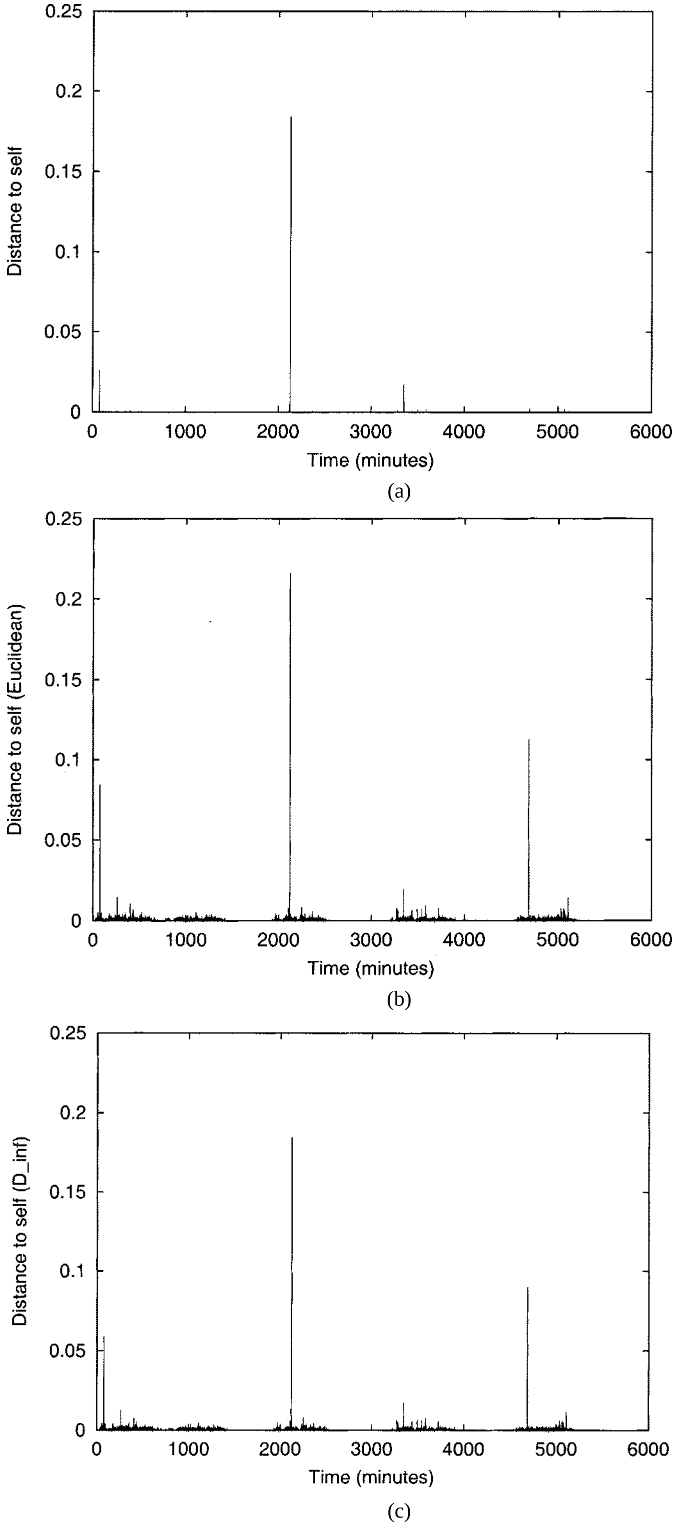
(a)

(b)

Fig. 2. Training and testing sets for the parameter number of packets per second. (a) Training (self) set corresponding to the first week. (b) Testing set corresponding to the second week.

2) A higher window size increases the sensitivity; this is reflected in the higher values of deviation.
3) A higher window size allows for the detection of temporal patterns. For the time series $T1$ and $T3$, increasing the window size does not modify the number of detected anomalies, but, for the time series $T2$, when the window size is increased from 1 [see Fig. 3(a)] to 3 [see Fig. 3(b) and (c)], one additional deviation (correspondent to attack 5) is detected. Clearly, this deviation was not caused by a value of this parameter (number of bytes per second) out of range; otherwise, it would be detected by the window size 1. There was a temporal pattern that was not seen in the training set and that might be the reason why it was reported as an anomaly.
4) The change of the distance metric from Euclidean [see Fig. 3(b)] to D_∞ [see Fig. 3(c)] does not modify the number and type of the deviations detected.

From the previous discussion, to detect the four attacks, it is necessary to take into account more than one parameter. In the following experiments, we used the three parameters to build the feature vector and test the ability of the system to detect the attacks. We performed two experiments varying the size of the sliding window:



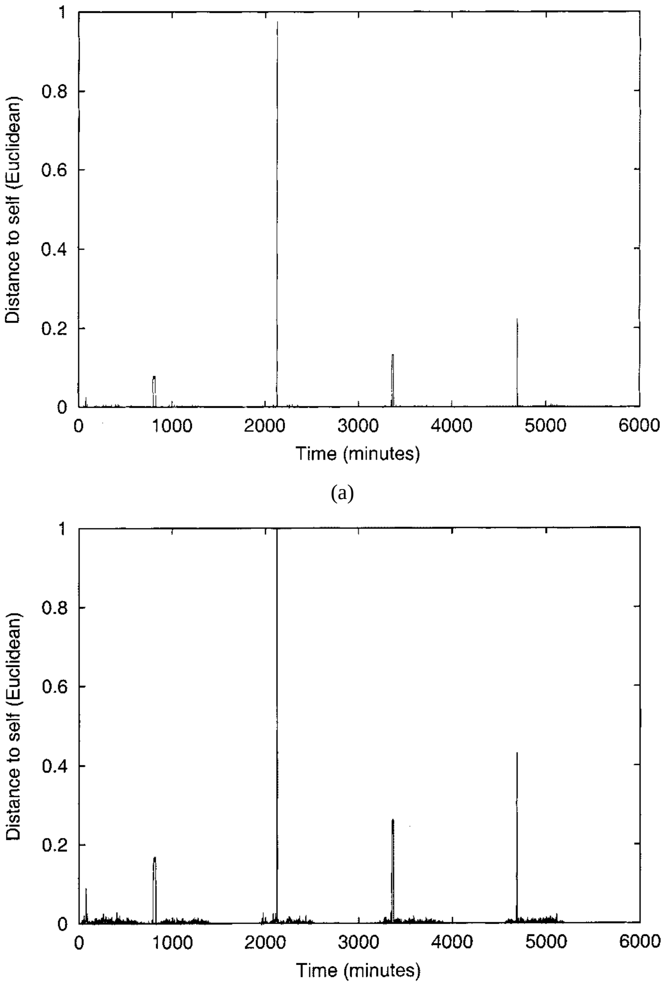
(a)

(b)

(c)

Fig. 3. Distance from the testing set $(T2)$ to the self set $(S2)$ $(\mu_{\text{nonself}}(\vec{x}))$. (a) Window size 1. Window size 3 for (b) Euclidean distance (c) D_∞ distance.

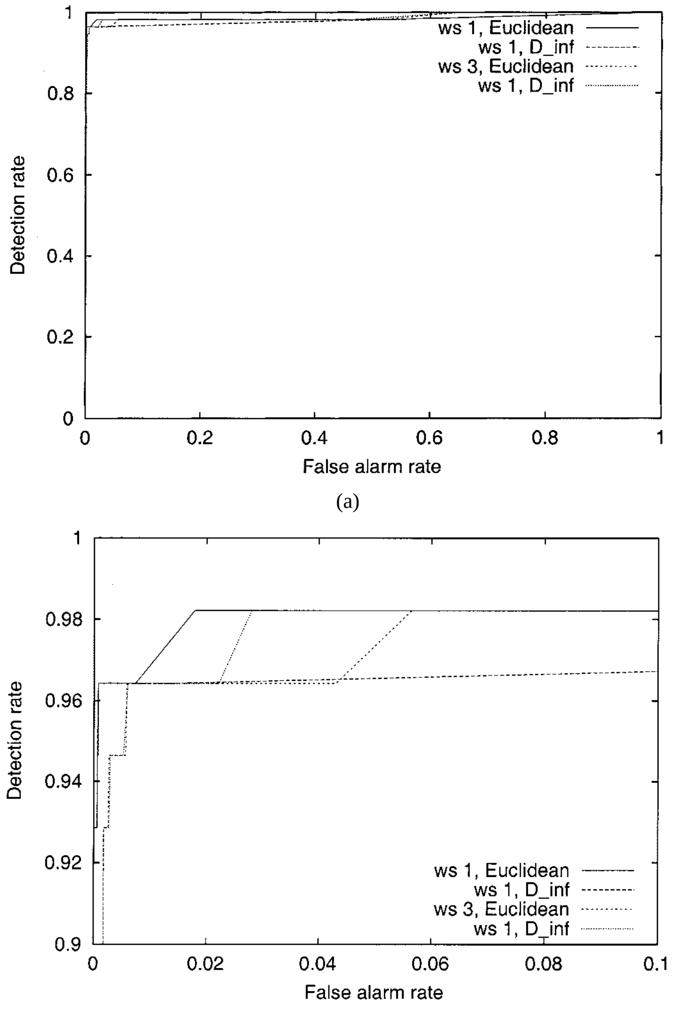
1) *Window size 1:* Feature vector structure $\{r_j^1, r_j^2, r_j^3\}$, where r_j^i, is taken from the time series Ti.
2) *Window size 3:* Feature vector structure $\{r_n^1, r_{n+1}^1, r_{n+2}^1, r_n^2, r_{n+1}^2, r_{n+2}^2, r_n^3, r_{n+1}^3, r_{n+2}^3\}$, where r_j^i, is taken from the time series Ti.



(a)

(b)

Fig. 4. Distance from test sets to the self set ($\mu_{\text{nonself}}(\vec{x})$) using $S1, S2, S3$. (a) Window size 1. (b) Window size 3.



(a)

(b)

Fig. 5. ROC diagrams for the $\mu_{\text{nonself}}(\vec{x})$ function shown on Fig. 4. (a) Full scale. (b) Detail of the upper left corner.

Fig. 4 shows the nonself characteristic function for features vectors conformed by samples of the three time series. In all the cases, there are five remarkable anomalies that correspond to the five attacks. Like the previous experiments, an increase on the size of the window increases the sensitivity of the anomaly detection function. However, this could generate more false positives. In order to measure the accuracy of the anomaly detection functions, it is necessary to convert them to the crisp version. In this case, the output of the functions will be normal or abnormal. This output can be compared with the information of the attacks to calculate how many anomalies (caused by an attack) were detected accurately.

According to Definition 3, the crisp version of the anomaly detection function $\mu_{\text{non\_self}}(\vec{x})$ is generated by specifying a threshold t. This threshold indicates the frontier between normal and abnormal. Clearly, the value of t will affect the capabilities of the system to detect accurately. A very large value of t will allow large variability on the normal (self), increasing the rate of false negatives; a very small value of t will restrict the normal set causing an increase on the number of detections, but also increase the number of false positives (false alarms). In order to show this tradeoff between false alarm rate and detection rate, receiver operating characteristics (ROCs) [28] diagrams are drawn. The anomaly detection function $\mu_{\text{nonself},t}(\vec{x})$ is tested with different values of t, the detection and false alarm rates are calculated, and this generates a set of points that constitutes the ROC diagram.

Fig. 5 shows ROC diagrams for the $\mu_{\text{nonself}}(\vec{x})$ functions shown in Fig. 4. In general, the behavior of the four functions is very similar: high detection rates with a small false alarm rate. The anomaly detection functions that use window size 3 show a slightly better behavior in terms of detection rates. This could be attributed to the higher sensitivity, produced by a larger window, to temporal patterns. However, this causes more false alarms. We think it is explained by the fact that after an attack, some disturbance is still present in the system and the function with larger window size is able to detect it.

The PC technique has shown to work well on the performed experiments. The main drawback of this technique is its memory requirements, since it is necessary to store all the samples that constitute the normal profile. The amount of data generated by network traffic can be large and this can make this approach unfeasible. This is the main motivation for the NC approach, compressing the information of the normal profile without a significative loss of accuracy.

III. NC Approach

A. Background and Previous Works

The main role of the immune system is to distinguish between self (all cells or molecules in the body) and nonself (others). The nonself elements are further categorized in order to determine specific response for protection and recovery from different diseases. In particular, the self–nonself discrimination is achieved in part by T cells, which have receptors on their surface that can detect foreign proteins (antigens). During the generation of T cells, receptors are made by a pseudorandom genetic rearrangement process [11], [14], [29]. Then, they undergo a censoring process, called negative selection, in the thymus, where T cells that react against self-proteins are destroyed, so only those that do not bind to self-proteins are allowed to leave the thymus. These matured T cells then circulate throughout the body to perform immunological functions to protect against foreign antigens. Rather than relying on a central controller, the immune system, however, uses a distributed detection and response mechanism for their survival from foreign invaders.

Forrest *et al.* [16] developed a negative selection algorithm based on the principles of self–nonself discrimination in the immune system. The negative-selection algorithm can be summarized as follows.

1) Define self as a collection of strings of length l over a finite alphabet, a collection that needs to be monitored. For example, S may be a normal pattern of activity, which is segmented into equal-sized substrings.
2) Generate a set R of detectors, each of which fails to match any string in S.
3) Monitor S for changes by continually matching the detectors in R against S. If any detector ever matches, then a change is known to have occurred, as the detectors are designed to not match any of the original strings in S.

There are many versions of the algorithm with varying degrees of computational complexities to generate detectors efficiently; they used a binary representation and run in linear time with the size of self, dependent on the choice of matching rule [11], [14], [16], [19], [29].

B. Extension of Negative Selection Approach

Instead of using binary encoding in the negative selection algorithm [16], our approach uses real-valued representation to characterize the Self–Nonself space and evolve a set of detectors that can cover the (Nonself) complementary subspace (as shown in Fig. 6). The basic structure of these detector rules is as follows:

$$\begin{array}{lll} R^1: \text{If} & \text{Cond}_1, & \text{then } \textbf{nonself} \\ \vdots & \vdots & \vdots \\ R^m: \text{If} & \text{Cond}_m, & \text{then } \textbf{nonself} \end{array}$$

where

Cond_i = $x_1 \in [\text{low}_1^i, \text{high}_1^i]$ and... and $x_n \in [\text{low}_n^i, \text{high}_n^i]$;

$(x_1, \ldots, x_n)$ feature vector;

$[\text{low}_i^j, \text{high}_i^j]$ lower and upper values for the feature x_i in the condition part of the rule R^j.

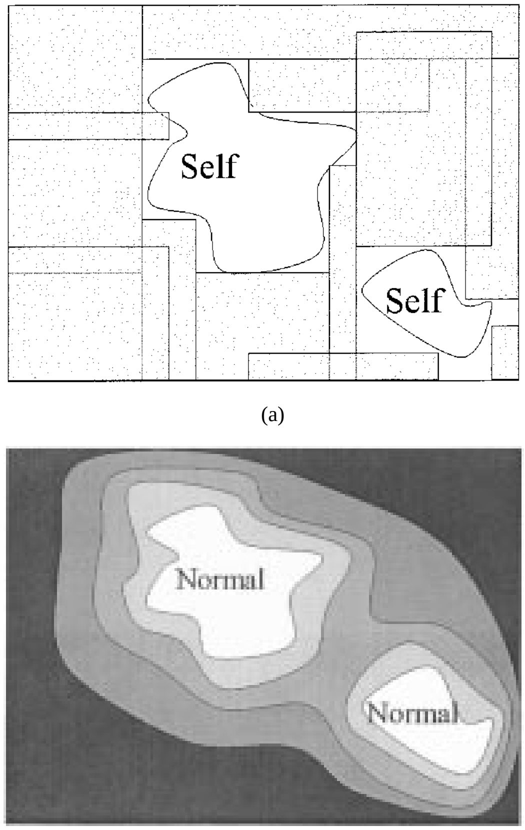


Fig. 6. (a) Approximation of the nonself space by rectangular interval rules. (b) Levels of deviation from the normal in the nonself space.

The condition part of each rule defines a hypercube in the descriptor space ($[0.0, 1.0]^n$). Then, a set of these rules tries to cover the nonself space with hypercubes. For the case $n = 2$, the condition part of a rule represents a rectangle. Fig. 6(a) illustrates an example of this kind of cover for $n = 2$.

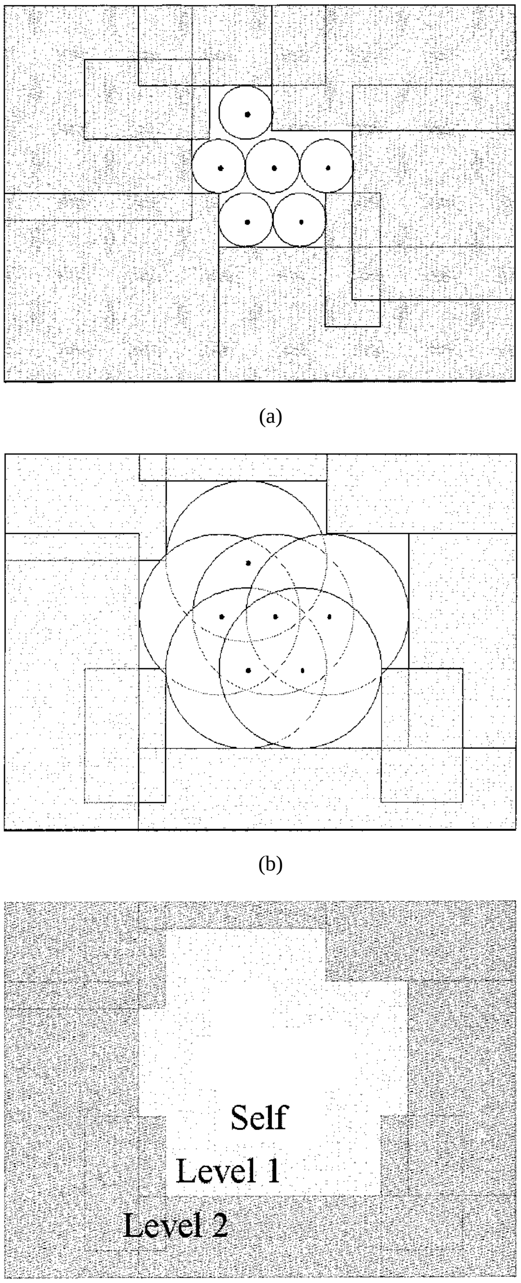
The nonself characteristic function (crisp version) generated by a set of rules $R = \{R^1, \ldots, R^m\}$ is defined as follows:

$$\chi_{\text{nonself},R}(\vec{x}) = \begin{cases} 1, & \text{if } \exists R^j \in R \text{ such that } \vec{x} \in R^j \\ 0, & \text{otherwise} \end{cases}.$$

We used a genetic algorithm (GA) to evolve rules to cover the nonself space. These rules constitute the complement of the normal values of the feature vectors. A rule is considered good if it does not cover positive samples and its area is large. These criteria guide the evolution process performed by the GA.

As was described previously, a good characterization of the abnormal (nonself) space should be noncrisp. Then, the nonself space is further divided in different levels of deviation. In Fig. 6(b), these levels of deviation are shown as concentric regions around the self zones.

In order to characterize the different levels of abnormality, we considered a variability parameter (called v) to the set of normal descriptors samples, where v represents the level of variability that we allow in the normal (self) space. A higher value of v means more variability (a larger self space); a lower



(a)

(b)

(c)

Fig. 7. Set of normal samples is represented as points in a 2-D space. Circle around each sample point represents the allowable deviation. (a) Rectangular rules cover the nonself (abnormal) space using a small value of v. (b) Rectangular rules cover the nonself space using a large value of v. (c) Level of deviation defined by each v, where level 1 corresponds to nonself cover in (a) and level 2 corresponds to nonself cover in (b).

value of v represents less variability (a smaller Self space). Fig. 7 shows two sets of rules that characterize self spaces with a large and small value of v. Fig. 7(a) shows a covering using a small variability parameter v. Fig. 7(b) shows a covering using a larger value of v. The variability parameter can be assumed as the radius of a hypersphere around the self samples. Fig. 7(c) shows the levels of deviation defined by the two coverings.

In the nonself space, we use a GA with different values of v to generate a set of rules that can provide complete coverage. A set of rules looks like

$$\begin{array}{rll} R^1 : \text{If} & \text{Cond}_1, & \text{then } \mathbf{Level\ 1} \\ \vdots \ \vdots & & \vdots \\ R^i : \text{If} & \text{Cond}_i, & \text{then } \mathbf{Level\ 1} \\ R^{i+1} : \text{If} & \text{Cond}_{i+1}, & \text{then } \mathbf{Level\ 2} \\ \vdots \ \vdots & & \vdots \\ R^j : \text{If} & \text{Cond}_j, & \text{then } \mathbf{Level\ 2} \\ \vdots \ \vdots & & \vdots \end{array}.$$

The different levels of deviation are organized hierarchically such that level 1 contains level 2, level 2 contains level 3, and so forth. This means that a descriptor can be matched by more than one rule, but the highest level reported will be assigned. This set of rules generates a noncrisp characteristic function for the nonself space

$$\mu_{\text{nonself}}(\vec{x}) = \max(\{l \mid \exists R^j \in R, \vec{x} \in R^j \text{ and } l = \text{level}(R^j)\} \cup \{0\})$$

where $\text{level}(R^j)$ represents the deviation level reported by the rule R^j.

C. Genetic Algorithm in Detection Rule Generation

The GA, attempts to evolve "good" rules [6], [7], [12] that cover the nonself space. The goodness of a rule is determined by various factors: the number of normal samples that it covers, its area, and the overlapping with other rules. This is clearly a multiobjective multimodal optimization problem. We are not interested in one solution, but a set of solutions that collectively can solve the problem (covering of the nonself region).

A niching technique is used with GAs to generate different rules. The input to the GA is a set of feature vectors $S = \{x^1, \ldots, x^m\}$, which represents samples of the normal behavior of the parameter. Each element x^j in S is a n-dimensional vector $x^j = (x_1^j, \ldots, x_n^j)$.

The algorithm for rule generation is shown in Fig. 8, where

S	normal descriptors training set;
v	level of variability;
maxRules	maximum number of rules in the solution set;
minFitness	minimum allowed for a rule to be included in the solution set;
maxAttempts:	maximum number of attempts to try to evolve a rule with a fitness greater or equal to *minFitness*.

Each individual (chromosome) in the GA represents the condition part of a rule, since the consequent part is the same for all the rules (the descriptor belongs to nonself). However, the levels of deviation in nonself space are considered by the variability factor (v). Each element of the chromosome is represented by a fixed number of bits (in our case, 8 bits). In order to decode the chromosome, the string of bits is mapped to the interval [0.0, 1.0].

1) Fitness Evaluation: Given a rule R with condition part $(x_1 \in [\text{low}_1, \text{high}_1]\ \mathbf{and} \ldots \mathbf{and}\ x_n \in [\text{low}_n, \text{high}_n])$, we say that a feature vector $x^j = (x_1^j, \ldots, x_n^j)$ satisfies the rule (represented for $x^j \in R$) if the hypersphere with center x^j and radius v intercepts the hyperrectangle defined by the points $(\text{low}_i, \ldots, \text{low}_n)$ and $(\text{high}_1, \ldots, \text{high}_n)$.

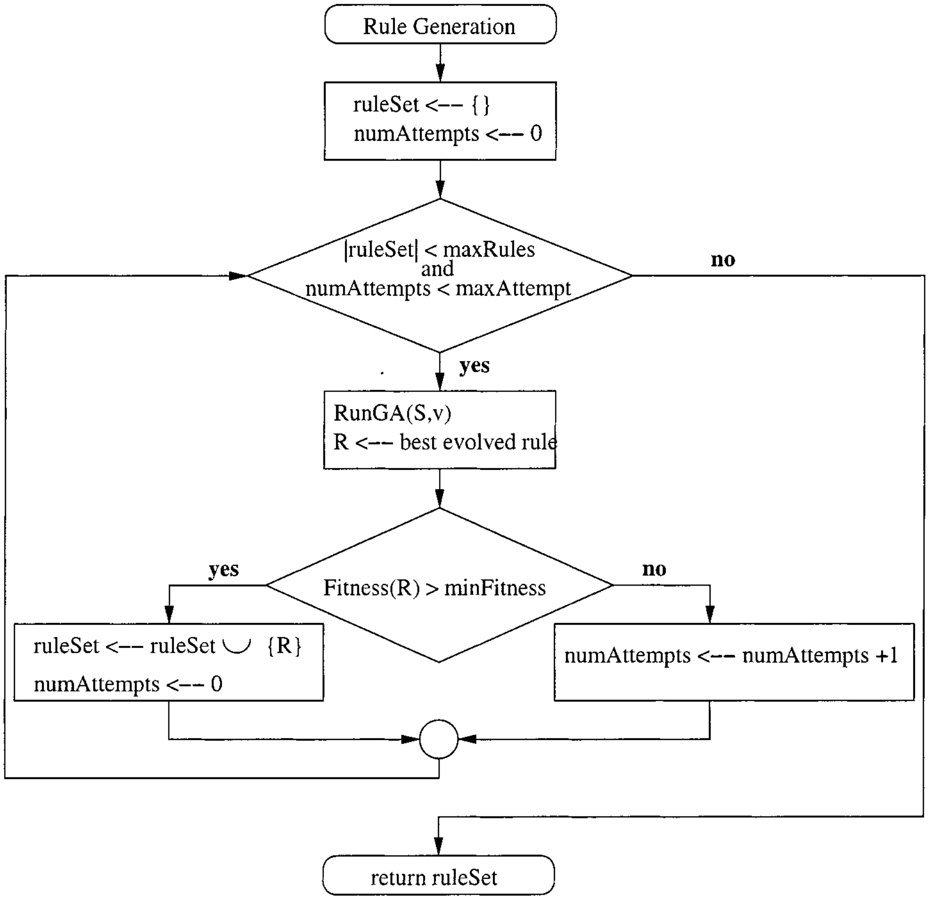


Fig. 8. NC rule generation using a GA.

The raw fitness of a rule is calculated taking in account the following two factors:

1) the number of elements in the training set S that belongs to the subspace represented by the rule

$$\mathrm{num\_elements}(R) = \{x^i \in S \,|\, x^i \in R\};$$

2) the volume of the subspace represented by the rule

$$\mathrm{volume}(R) = \prod_{i=1}^{n} (\mathrm{high}_i - \mathrm{low}_i).$$

The raw fitness is defined as

$$\mathrm{raw\_fitness}_R = \mathrm{volume}(R) - C \cdot \mathrm{num\_elements}(R)$$

where C is the coefficient of sensitivity. It specifies the amount of penalization that a rule suffers if it covers normal samples. The bigger the coefficient, the higher the penalty value. The raw fitness can take negative values also.

Since a covering of the nonself space is accomplished by a set of rules, it is necessary to evolve multiple rules. In order to evolve different rules a sequential niching algorithm is applied.

2) Sequential Niching Algorithm: The idea is to run the GA multiple times [1] to generate different rules so as to cover the entire nonself region. In each run, we want to generate new rules, that is, rules that cover the rest of the nonself region. The raw fitness of each rule is modified according to the overlap with the previously chosen rules. The following pseudocode segment shows how the final fitness of the rule R is calculated.

```
$\mathrm{fitness}_R \leftarrow \mathrm{raw\_fitness}_R$
for each $R^j \in \mathrm{ruleSet}$ do
  $\mathrm{fitness}_R \leftarrow \mathrm{raw\_fitness}_R - \mathrm{volume}(R \cap R^j)$
end-For
```

volume( ) calculates the volume of the subspace specified in the argument.

D. Experimentation and Results

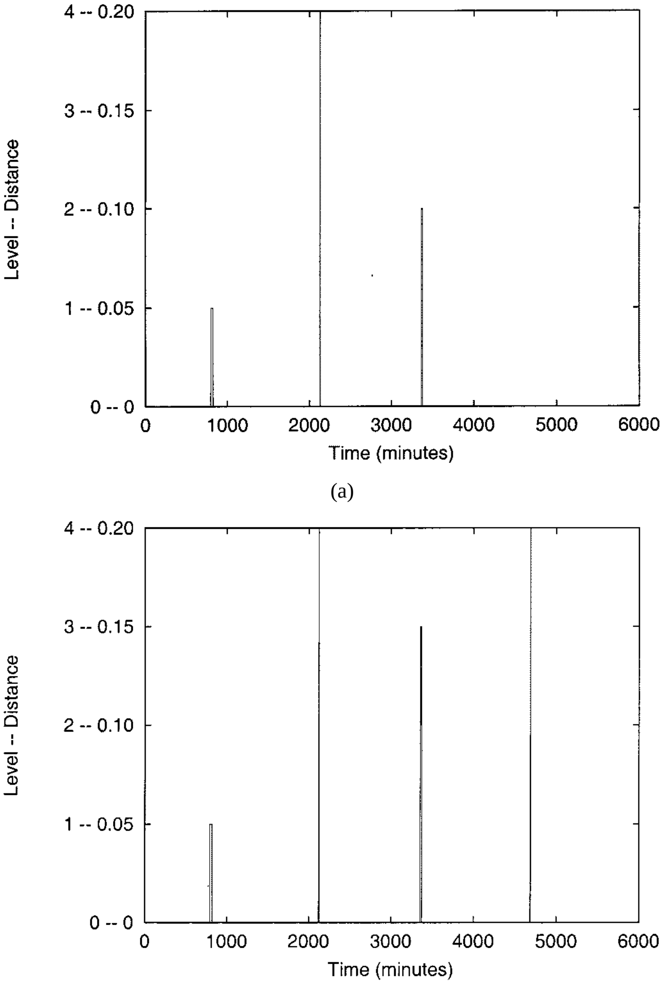
In order to test the NC approach, we used the data sets from [21], as in Section II-A. We used as training set the time series $S1, S2$, and $S3$ and as testing set the time series $T1, T2$, and $T3$, with a window size of 3 and 1, respectively (the time series are described in Table II).

The parameters for the GA were the following: population size 100, number of generations 1500, mutation rate 0.2, crossover rate 1.0, and coefficient of sensitivity 1.0 (high sensitivity).

The GA was run with variability parameter (v) equal to 0.05, 0.1, 0.15, and 0.2, respectively. Then, the elements in the testing set were classified using rules generated for each level (different value of v). This process was repeated ten times and the results reported correspond to the average of these runs.

TABLE III
NUMBER OF GENERATED RULES FOR EACH DEVIATION LEVEL

Level	Radius	Avg. Num. Rules (Window size = 1)	Avg. Num. Rules (Window size = 3)
1	0.05	1.1	19.5
2	0.1	1.1	20.7
3	0.15	1	26
4	0.2	1.1	28



(a)

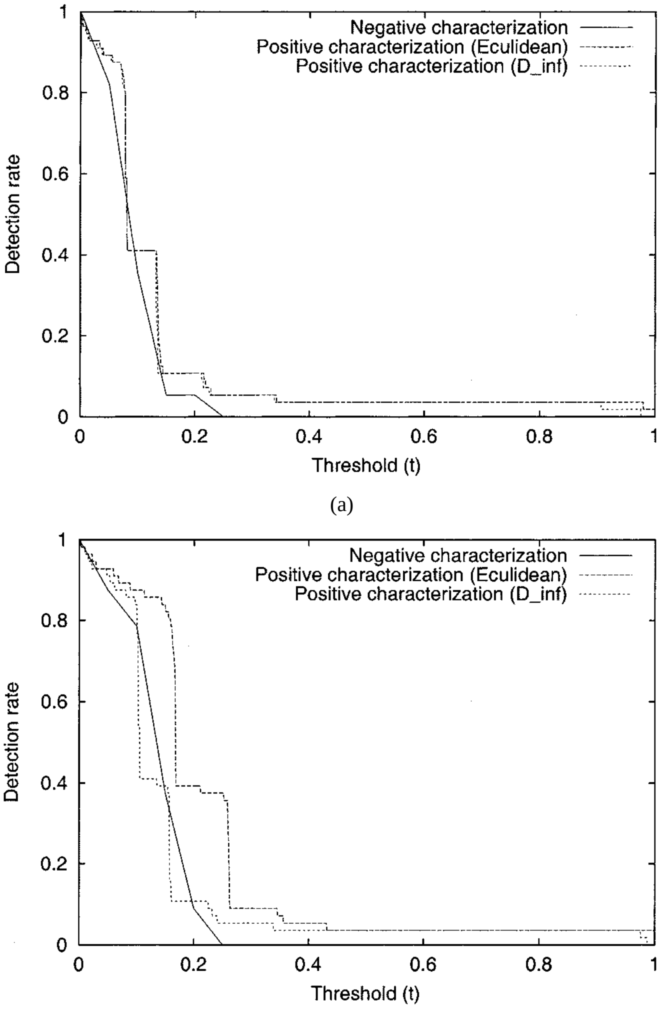
(b)

Fig. 9. Indicates the deviations in the testing set detected by evolved rule set. (a) Window size 1. (b) Window size 3.

Table III shows the number of rules generated by the GA for each level. There is a clear difference between the number of rules when the window size changes: the number of rules changes with the window size as the pattern space becomes larger.

Fig. 9 shows two typical attack profiles produced by the application of the genetic generated rules to the testing set. With a window size of 1, three of five attacks are detected, while with a window size of 3, four of five attacks are detected.

NC is clearly more efficient (in time and space) compared to the PC. In the case of a window size of 1, the PC needs to store $5202 * 3 = 15\,606$ floating-point values; the NC only has to store $4 * 6 = 24$ floating-point values, so the compression ratio is approximately 1000:1.5. In the case of the window size of 3, the ratio is 46 728:1 698,[5] approximately 100:8. It seems to be a tradeoff between compactness of the rule set representation and accuracy. Validity of these arguments is observed on our results. Fig. 10 shows how the rate of true positives (detection rate) changes according to the value of the threshold t. In both cases, The PC technique has better performance than NC, but only by a small difference. In general, the NC technique shows detection rates similar to the more accurate (but more expensive) PC technique. Table IV summarizes the best true positive rates (with a maximum false alarm of 1%) accomplished by the two techniques.



(a)

(b)

Fig. 10. Comparison of the true positives rate of the detection function $\mu_{\text{nonself},t}(\vec{x})$ generated by PC and NC for different values of t. (a) Window size 1. (b) Window size 3.

The results in Table V suggest that the NC approach better approximates the deviation reported by PC using D_∞ distance. To support this claim in a more exact way, we measured the number of testing samples for the all possible differences between the PC reported level and the NC reported level. A difference of zero means that the reported levels are the same, a

[5] The number of floating point numbers needed by the PC is equal to $(5192 \text{ samples}) * (9 \text{ dimensions}) = 46\ 728$. The number of floating points numbers needed by the NC is $(94 \text{ rules}) * (18 \text{ floating values per rule}) = 1\ 698$.

TABLE IV
BEST TRUE POSITIVE RATES FOR THE DIFFERENT TECHNIQUES WITH A MAXIMUM FALSE ALARM RATE OF 1%.

Detection Technique	**Window size 1**	**Window size 3**
Positive Characterization (Euclidean)	92.8%	96.4%
Positive Characterization (D_∞)	92.8%	92.8%
Negative Characterization	82.1%	87.5%

TABLE V
CONFUSION MATRIX FOR PC AND NC REPORTED DEVIATION

PC output level	**NC output level**				
	No deviation	Level 1	Level 2	Level 3	Level 4
Euclidean	[0.0,0.05]	[0.05,0.1]	[0.1,0.15]	[0.15,0.2]	[0.2,..]
[0.0,0.05]	5131	0	0	0	0
[0.05,0.1]	4	1	0	0	0
[0.1,0.15]	0	2.9	2.1	0	0
[0.15,0.2]	0	22	2	0	0
[0.2,..]	0	0	6.9	10.5	9.6
D_∞					
[0.0,0.05]	5132	0	0	0	0
[0.05,0.1]	3	7.8	0.2	0	0
[0.1,0.15]	0	18.1	3.9	0	0
[0.15,0.2]	0	0	6.9	9.5	0.6
[0.2,..]	0	0	0	1	9

Values of the matrix elements correspond to the number of testing samples in each class. Diagonal values represent correct classification.

TABLE VI
DIFFERENCE BETWEEN PC AND NC REPORTED LEVELS FOR TEST DATA SET

Difference betweenPC and NC reported level	**Euclidean distance**	**D_∞ distance**
0	20.8%	50.3%
1	31.8%	49.7%
2	47.3%	0.0%
3	0.0%	0.0%
4	0.0%	0.0%

It is expressed as a percentage of the abnormal feature vectors (distance greater than 0.05). A difference of zero means that the level reported by PC and NC are the same, a difference of one means that the results differ by 1 level, etc.

difference of one means that the results differ in one level, etc. The results for the two distances and two windows size are reported in Table VI.

The results are very different when the different distances are used for the PC algorithm. Clearly, when the D_∞ distance is used in the PC, the results of the comparison improve. Despite the fact that only 50.3% of the outputs from the NC algorithm are the same to the PC algorithm, 100% of the NC outputs are in the range of zero or one level of difference from the PC. The distance metric determines the structure of a metric space. For instance, in a Euclidean space, the set of points that are at the same distance from a fixed point corresponds to a circle (a hypersphere in higher dimensions). In the D_∞ metric space, this set of points corresponds to a rectangle (hyperrectangle). Therefore, the rectangular rules used by the NC approach are better suited to approximate the structure of the D_∞ metric space and this is reflected in the experimental results.

IV. CONCLUSION

As was mentioned before, the proposed NC technique produces a good estimate of the level of deviation. In order to evaluate this estimate, a detailed comparison of the NC output level and PC distance range was performed. The results are illustrated in Table V in the form of a confusion matrix. For each element in the testing set, the function $\mu_{\text{nonself}}(\vec{x})$ generated by the NC is applied to determine the level of deviation. This level of deviation is compared with the distance range reported by the PC algorithm. Each row (and column) corresponds to a range or level of deviation. The ranges are specified on square brackets. A perfect output from the NC algorithm will generate only values in the diagonal. The immune system has the property that foreign invaders (i.e., the nonself) are recognized easily with few false positives. This process proceeds in two stages: in the first stage, the nonself is identified as a novelty (novelty detection) and the immune system then preserves a long-term memory for this pattern. By understanding the dynamics of the immune system, it is possible to implement a pattern recognition mechanism in the complement space where false positives and false negatives can be traded off as shown by ROC curves.

We investigated an immunocomputing technique to evolve novel pattern detectors in the complement pattern space to identify any changes in the normal behavior of monitored behavior patterns. This technique (NC) is used to characterize and identify different intrusive activities by monitoring network traffic and compared with another approach (PC). We used a real-world data set [21] that has been used by other researchers for testing different approaches. The following are some preliminary observations.

1) When PC and NC approaches are compared, PC appears to be more precise, but it requires more time and space resources. NC is less precise, but requires fewer resources.
2) Results demonstrate that the NC approach to detector generation is feasible. It was able to detect four of the five attacks detected by the PC (with a detection rate of 87.5% and a maximum false alarms rate of 1%), while only using 10% of the space.
3) The best results were produced when we used a window size of 3. We observed that a larger window size makes the system more sensitive to deviations.

Our future research includes the testing of different covering strategies of the nonself space (for instance, using hyperspheres), experimentation with other intrusion detection data sets (both online and offline), and the development of new algorithms to generate nonself covering rules.

References

[1] D. Beasley, D. R. Bull, and R. R. Martin, “A sequential niche technique for multimodal function optimization,” *Evol. Comput.*, vol. 1, no. 2, pp. 101–125, 1993.

[2] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[3] ——, “K-d trees for semidynamic point sets,” in *Proc. 6th Annu. ACM Symp. Computational Geometry*, May 1990, pp. 187–197.

[4] J. Cannnady, “Next generation intrusion detection: Autonomous reinforcement learning of network attacks,” in *Proc. 23rd Nat. Information Systems Security Conf.*, Oct. 2000, pp. 1–12.

[5] M. Crosbie and E. H. Spafford, “Applying genetic programming to intrusion detection,” in *Working Notes for the AAAI Symposium on Genetic Programming*, E. V. Siegel and J. R. Koza, Eds. Cambridge, MA: MIT Press, 1995, pp. 1–8.

[6] D. Dagupta and F. Gonzalez, “Evolving complex fuzzy classifier rules using a linear genetic representation,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. San Mateo, CA: Morgan Kaufmann, 2001.

[7] ——, “An intelligent decision support system for intrusion detection and response,” in *Information Assurance in Computer Networks*. Berlin, Germany: Springer-Verlag, 2001, Lecture Notes in Computer Science, pp. 1–14.

[8] D. Dasgupta, *Artificial Immune Systems and Their Applications*. New York: Springer-Verlag, 1999.

[9] ——, “Immunity-based intrusion detection system: A general framework,” in *Proc. 22nd Nat. Information Systems Security Conf.*, Oct. 1999, pp. 147–160.

[10] ——, “Mobile security agents for network traffic analysis,” in *Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX-II)*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1999.

[11] D. Dasgupta and S. Forrest, “Novelty detection in time series data using ideas from immunology,” in *Proc. Int. Conf. Intelligent Systems*, June 1996, pp. 87–92.

[12] D. Dasgupta and Z. Michalewicz, Eds., *Evolutionary Algorithms in Engineering Applications*. New York, NY: Springer-Verlag, 1997.

[13] D. Denning, “An intrusion-detection model,” *IEEE Trans. Software Eng.*, vol. 13, pp. 222–232, Feb. 1987.

[14] P. D’haeseleer, S. Forrest, and P. Helman, “An immunological approach to change detection: Algorithms,” in *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1996, pp. 110–119.

[15] E. Eskin, “Anomaly detection over noisy data using learned probability distributions,” in *Proceedings of the 17th International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann, 2000, pp. 255–262.

[16] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri, “Self-nonself discrimination in a computer,” in *Proc. IEEE Symp. Research in Security and Privacy*, May 1994, pp. 202–212.

[17] J. Frank, “Artificial intelligence and intrusion detection: Current and future directions,” in *Proc. 17th Nat. Computer Security Conf.*, Oct. 1994, pp. 22–33.

[18] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 209–226, 1977.

[19] S. A. Hofmeyr and S. Forrest, “Architecture for an artificial immune system,” *Evol. Comput.*, vol. 8, no. 4, pp. 443–473, 2000.

[20] J. O. Kephart, “A biologically inspired immune system for computers,” in *Proceedings of Artificial Life IV*, R. Brooks and P. Maes, Eds. Cambridge, MA: MIT Press, 1994, pp. 130–139.

[21] DARPA Intrusion Detection Evaluation (1999). [Online]. Available: http://www.ll.mit.edu/IST/ideval/index.html

[22] T. Lane, “Machine learning techniques for the computer security,” Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 2000.

[23] T. Lane and C. Brodley, “Temporal sequence learning and data reduction for anomaly detection,” *ACM Trans. Info. Syst. Security*, no. 2, pp. 295–331, 1999.

[24] T. Lane and C. E. Brodley, “Data reduction techniques for instance-based learning from human/computer interface data,” in *Proceedings of the 17th International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann, 2000, pp. 519–526.

[25] W. Lee and S. Stolfo, “Data mining approaches for intrusion detection,” in *Proc. 7th USENIX Security Symp.*, Jan. 1998, pp. 26–29.

[26] W. Lee and D. Xiang, “Information-theoretic measures for anomaly detection,” in *Proc. 2001 IEEE Symp. Security and Privacy*, May 2001, pp. 14–16.

[27] D. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. presented at Proc. 2nd Annu. CGC Workshop on Computational Geometry. [Online]. Available: http://www.cs.umd.edu/mount/ANN

[28] F. Provost, T. Fawcett, and R. Kohavi, “The case against accuracy estimation for comparing induction algorithms,” in *Proceedings of 15th International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1998, pp. 445–453.

[29] A. Somayaji, S. Hofmeyr, and S. Forrest, “Principles of a computer immune system,” in *Proc. 2nd New Security Paradigms Workshop*, Sept. 1997, pp. 75–82.

Dipankar Dasgupta (S’95–A’96) received the Ph.D. degree in computer science from the University of Strathclyde, Glasgow, U.K., in 1994.

He is currently a Faculty Member with the Computer Science Division of the Mathematical Sciences Department, University of Memphis, Memphis, TN. His current research interests include scientific computing, tracking real-world problems through interdisciplinary cooperation, artificial intelligence, genetic algorithms, neural networks, immunocomputing, and their applications. He serves as a program committee member in many international conferences, has organized special tracks and workshops on artificial immune systems, and offers tutorials on the topics at international conferences since. He has authored or coauthored more than 70 papers in book chapters, journals, and international conferences, and has edited the book *Artificial Immune Systems and Their Applications* (New York: Springer-Verlag, 1999).

Dr. Dasgupta is a Member of the ACM.

Fabio González (S’02) received the B.S. degree in systems engineering and the M.Sc. degree in mathematics from the National University of Colombia, Bogota, Columbia, in 1993 and 1998, respectively. He is working toward the Ph.D. degree in computer science at the University of Memphis, Memphis, TN.

He has been an Assistant Professor with the Department of Systems Engineering, National University of Colombia, since 1999. He is also a Research Assistant with the Computer Science Division, Mathematical Sciences department, University of Memphis. His current research interests include evolutionary computation, artificial immune systems, and bio-inspired computation and its applications.