AU3045_C007.fm Page 165 Saturday, June 25, 2005 9:06 AM

Chapter 7

Artificial Immune Systems in Intrusion Detection

Dipankar Dasgupta and Fabio Gonzalez

7.1 Introduction

The biological immune system (BIS) is a complex network of specialized tissues, organs, cells, and chemicals. Its main function is to recognize the presence of strange elements in the body and respond to eliminate or neutralize the foreign invaders. All living organisms are exposed to many different microorganisms and viruses that are capable of causing illness. These microorganisms are called *pathogens*. In general, organisms try to protect against pathogens using different mechanisms including high temperature, low pH, and chemicals that repel or kill the invaders. More advanced organisms (vertebrates) have developed an efficient defense mechanism called the *immune system* [26]. Substances that can stimulate specific responses of the immune system are commonly referred to as *antigens* (pathogens usually act as antigens).

To be effective, the immune system must respond only to foreign antigens; therefore, it should be able to distinguish between the self (cells, proteins, and in general, any molecule that belongs to or is produced by the body) and the nonself (antigens) [7]. The self/nonself discrimination is an essential characteristic of the immune system because the outcome of an inappropriate response to self-molecules can be fatal.

AU3045_C007.fm Page 166 Saturday, June 25, 2005 9:06 AM

The immune system generates a large variety of cells and molecules for defensive purposes. These cells and molecules interact with each other and form a dynamic network of active immune cells while detecting and eliminating antigens. It is difficult to give a concise picture of such a complex system, as many of the mechanisms are not completely understood. Detailed review of the natural immune system and its functionalities may be found elsewhere [26,27,33].

7.1.1 Multilayered Protection

The immune system can be envisioned as a multilayer system with defense mechanisms in several layers [24]. The three main layers include the anatomic barrier, innate immunity, and adaptive immunity. They are described as follows:

- *Anatomic barrier:* The first layer is the anatomic barrier, composed of the skin and the surface of mucous membranes. Intact skin prevents the penetration of most pathogens and also inhibits most bacterial growth because of its low pH. On the other hand, many pathogens enter the body by binding or penetrating through the mucous membranes; these membranes provide a number of nonspecific mechanisms that help to prevent such entry. Saliva, tears, and some mucous secretions act to wash away potential invaders and also contain antibacterial and antiviral substances [33].
- *Innate immunity:* Innate immunity [26], which is also known as nonspecific immunity, refers to the defense mechanism against foreign invaders that individuals are born with. Innate immunity is mainly composed of the following mechanisms:
  - *Physiologic barriers:* This includes mechanisms such as temperature, pH, oxygen tension, and various soluble chemicals. The purpose of these mechanisms is to provide detrimental living conditions for foreign pathogens. For instance, the low acidity of the gastric system acts as a barrier to infection by ingested microorganisms because they cannot survive the low pH of the stomach.
  - *Phagocytic barriers:* Some specialized cells (such as macrophages, neutrophils, and natural killer cells) are able to ingest specific material, including whole pathogenic microorganisms. This ingestion has two purposes: to kill the antigen and to present fragments of the invader's proteins to other immune cells and molecules.
  - *Inflammatory response:* Activated macrophages produce proteins called *cytokines*. They work as hormone-like messengers

AU3045_C007.fm Page 167 Saturday, June 25, 2005 9:06 AM

that induce the inflammatory response, which is characterized by vasodilatation and rise in capillary permeability. These changes allow a large number of circulating immune cells to be recruited to the site of the infection. Cytokines are also produced by other immune cells and nonimmune cells, for example, those that secrete cytokines when damaged [26].

- *Adaptive immunity:* This layer is described in detail in the following subsections.

7.1.2 Adaptive Immunity

It is also called acquired or specific immunity, which represents the part of the immune system that is able to specifically recognize and selectively eliminate foreign microorganisms and molecules. It is important to note that acquired immunity does not act independently of innate immunity; on the contrary, they work together to eliminate foreign invaders. For instance, phagocytic cells (innate immunity) are involved in the activation of adaptive immune response. Also, some soluble factors, produced during a specific immune response, have been found to augment the activity of these phagocytic cells [33].

7.1.2.1 Characteristics of Adaptive Immunity

An important part of the adaptive immune system is managed by white blood cells, called *lymphocytes*. These cells are produced in the bone marrow, circulate in the blood and lymph system, and reside in various lymphoid organs to perform immunological functions.

- *B-cells and T-cells:* They represent the major population of lymphocytes. These cells are produced in the bone marrow and are inert initially, i.e., they are not capable of executing their functions. To become immune competent, they have to go through a maturation process. In the case of B-cells, the maturation process occurs in the bone marrow itself. For T-cells, they have to migrate first to the thymus, where they mature. In general, a mature lymphocyte can be considered as a detector that can detect specific antigens. There are billions of these detectors that circulate in the body, constituting an effective, distributed anomaly detection and response system [33].
- *Humoral immunity:* Mature B-cells express unique antigen-binding receptors (ABR) on their surface. The interaction of ABR with specific antigen induces proliferation and differentiation of B-cells

AU3045_C007.fm Page 168 Saturday, June 25, 2005 9:06 AM

into antibody-secreting plasma cells. An antibody is a molecule that binds to antigens and neutralizes them or facilitates their elimination. Antigens coated with antibodies can be eliminated in multiple ways: by phagocytic cells, by the complement system, or by preventing them from performing any damaging functions (e.g., binding of viral particles to host cells) [40].

- *Cellular immunity:* During their maturation, T-cells express a unique ABR on their surface, called the T-cell receptor. Unlike B-cell ABR that can recognize antigens alone, T-cell receptors can only recognize antigenic peptides that are presented by cell-membrane proteins known as major histocompatibility complex (MHC) molecules. When a T-cell encounters antigens associated with an MHC molecule on a cell,* the T-cell proliferates and differentiates into memory T-cells and various effector T-cells. The cellular immunity is accomplished by these generated effector T-cells. There are different types of T-cells that interact in a complex way to kill altered self-cells (for instance, virus-infected cells) or to activate phagocytic cells [36].
- *Self/nonself discrimination:* The immune system can distinguish its own cells from foreign antigens, and so responds only to the dangerous nonself molecules. As was mentioned before, T-cells mature in the thymus. There, they go through a process of selection that ensures that they are able to recognize nonself peptides presented by MHC [7].
- *Negative selection:* The purpose of negative selection is to test for tolerance of self-cells. T-cells that recognize the combination of MHC and self-peptides fail this test. This process can be seen as a filtering of a large diversity of T-cells; only those T-cells that do not recognize self-peptides are retained [30].
- *Immune memory:* The immune system can "remember" a previous encounter with an antigen. This helps to deliver a quick response in subsequent encounters. In particular, immune-competent lymphocytes are able to recognize specific antigens through their ABR. The specificity of each T-cell and B-cell is determined prior to its contact with the antigen through random gene rearrangements in the bone marrow (or thymus) during the maturation process [32]. The presence of an antigen in the system and its subsequent interaction with mature lymphocytes trigger an immune response, resulting in the proliferation of lymphocytes with a unique antigenic

* In general, T-cells do not recognize whole antigen molecules; instead, their receptors detect fragments of the antigen called *peptides*, which are processed and presented by antigen-processing cells (APCs).

AU3045_C007.fm Page 169 Saturday, June 25, 2005 9:06 AM

specificity. This process of population expansion of particular T-cells and B-cells is called *clonal selection*. Clonal selection contributes to the specificity of adaptive immunity response because only lymphocytes whose receptors are specific to a given antigen will be cloned and, thus, mobilized for an immune response.

Another important consequence of clonal selection is immune memory [33]. The first encounter of naive immune-competent lymphocytes with an antigen generates the primary response, which, as discussed before, results in the proliferation of the lymphocytes that can recognize this specific antigen. Most of these lymphocytes die when the antigen is eliminated; however, some are kept as memory cells. The next occurrence of the same antigen can be detected quickly, activating a secondary response. This response is faster and more intense because of the availability of such memory cells.

7.1.3 Computational Aspects of the Immune System

From the point of view of information processing, the natural immune system exhibits many interesting characteristics. The following is a list of these characteristics [12,20]:

- *Pattern matching:* The immune system is able to recognize specific antigens and generate appropriate responses. This is accomplished by a recognition mechanism based on chemical binding of receptors and antigens. This binding depends on the molecular shape and on the electrostatic charge.
- *Feature extraction:* In general, immune receptors do not bind to the complete antigen but to peptides. In this way, the immune system can recognize an antigen just by matching segments of it. Antigenic feature is extracted (called peptides) and presented to the lymphocyte receptors by antigen-presenting cells (APC). These APCs act as filters that can extract the important information and remove the molecular noise.
- *Learning and memory:* The main characteristic of the adaptive immune system is that it is able to learn through interaction with the environment. The first time an antigen is detected, a primary response is induced that includes the proliferation of lymphocytes and a subsequent reduction. Some of these lymphocytes are kept as memory cells. The next time the same antigen is detected, the memory cells generate a faster and more intense response (secondary response). Memory cells work as an associative (highly) distributed memory.

AU: APC has been expanded earlier as "antigen-processing cells."

AU3045_C007.fm Page 170 Saturday, June 25, 2005 9:06 AM

- *Diversity:* The adaptive immune system can generate billions of different recognition molecules that are able to uniquely recognize different structures of foreign antigens. Clonal selection and hypermutation mechanisms constantly test different detector configurations for known and unknown antigens. This is a highly combinatorial process that explores the space of possible configurations for close-to-optimum receptors that can cope with the different types of antigens. Exploration is balanced with exploitation by favoring the reproduction of promising individuals.
- *Distributed processing:* Unlike the nervous system, the immune system does not possess a central controller. Detection and response can be executed locally and immediately without communicating with any central organ. This distributed behavior is accomplished by billions of immune molecules and cells that circulate in the blood and lymph systems and are capable of making decisions in a local collaborative environment.
- *Self-regulation:* Depending on the severity of the attack, the response of the immune system can range from very light and almost imperceptible to very strong. A stronger response uses a lot of resources to help repel the attacker. Once the invader is eliminated, the immune system regulates itself to stop the delivery of new resources and to release the used ones. Programmed cell death and clonal expansions are parts of this self-regulatory process.

7.2 Artificial Immune Systems

The study and design of artificial immune systems (AISs) is a relatively new area of research that tries to build computational systems that are inspired by the BIS [14]. There are many desirable computational features in the BIS that can be used to solve computational problems. In many respects, AISs are abstract computational models of the immune system; in fact, some AIS techniques are based on theoretical models of the BIS. However, the main difference lies in the use of AISs as a problem-solving technique.

A theoretical model that has served as a basis for some AISs is the idiotypic network theory proposed by Jerne [29]. This theory proposed that the BIS regulates itself by forming a network of B-cells that can enhance or suppress the expression of specific antibody types. This self-regulatory mechanism maintains a stable immune memory. The formation of such a network is only possible by the presence of paratopes on the B-cells that can be recognized by other B-cell epitopes. This recognition

AU3045_C007.fm Page 171 Saturday, June 25, 2005 9:06 AM

usually extends to more than one level, resulting in the formation of complex reaction networks. This model is a simplification of the BIS that ignores important elements such as T-lymphocytes and macrophages and concentrates on the modeling of the idiotypic networks.

Forrest and her group [21] proposed the negative-selection algorithm (NSA), which is inspired by the mechanism used by the immune system to train the T-cells to recognize antigens (nonself) and to prevent them from recognizing the body's own cells (self). Different variations of this algorithm have been applied to problems in anomaly detection [14,25], fault detection [13,41], and computer intrusion detection [15,21,24]. The rest of this chapter describes in detail the NSA, its versions, and applications to intrusion detection.

7.2.1 NSA

The immune system, however, can recognize and classify different novel patterns (pathogenic patterns of interest) and generate selective responses in nonself space. Self/nonself (or danger) discrimination may be one of the important tasks of the immune system during the process of pathogenic recognition.

This 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. Then they undergo a censoring process in the thymus, called *negative selection*, in which T-cells that react against self-proteins are destroyed; hence, 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. Forrest et al. [21] proposed the NSA based on self/nonself discrimination in the immune system.

The NSA is based on the principles of self/nonself discrimination in the immune system (Figure 7.1 shows the concept of self and nonself space). This can be summarized as follows (adopted from Reference 16):

AU: Should this be "adapted"?

- Define self as a collection S of elements in a feature space U, a collection that needs to be monitored. For instance, if U corresponds to the space of states of a system represented by a list of features, S can represent the subset of states that are considered normal for the system.
- Generate a set F of detectors, each of which fails to match any string in S. An approach that mimics the immune system generates random detectors and discards those that match any element in

AU3045_C007.fm Page 172 Saturday, June 25, 2005 9:06 AM

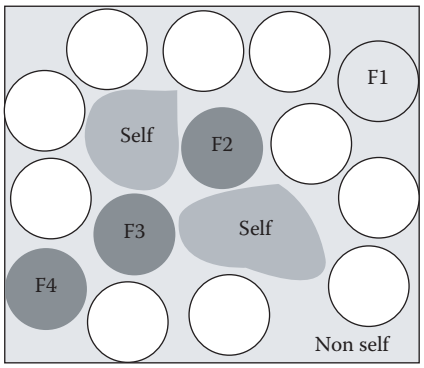


Figure 7.1 Conceptual view of self and nonself. Here, F1, F2, and F3 indicate different known attack types.

the self set. However, a more efficient approach [17] tries to minimize the number of generated detectors while maximizing the coverage of the nonself space.

- Monitor *S* for changes by continually matching the detectors in *F* against *S*. If any detector ever matches, then a change is known to have occurred, as the detectors are designed not to match any representative samples of *S*.

This description is very general and does not say anything about the representation of the problem space and the type of matching rule that is used. It is, however, clear that the algorithmic complexity of generating good detectors can vary significantly, which depends on the type of problem space (continuous, discrete, mixed, etc.), detector-encoding scheme, and the matching rule (which determines if a detector matches an element or not). Most of the past works on the NSA had been restricted to the binary matching rules like *r*-contiguous, hamming distance, r-chunk, etc. The primary reason for this choice is the ease of use, and there exist efficient algorithms that exploited the properties of the binary representation and its matching rules [17]. However, there are practical issues that prevent the binary NSA from being applied more extensively:

- Scalability is one such issue. To guarantee good levels of detection, a large number of detectors have to be generated (depending on the size of the self). For some problems, the number of detectors could be unmanageable.
- The low-level detector representation prevents the extraction of meaningful domain knowledge. This makes it difficult to analyze reasons for reporting an anomaly in a monitored system or process.

AU3045_C007.fm Page 173 Saturday, June 25, 2005 9:06 AM

- A sharp distinction exists between the normal and abnormal. This divides the space into two subsets: self (the normal) and the nonself (abnormal). An element in the space is considered to be abnormal if there exists a detector that matches it. In reality, normalcy is not a crisp concept. A natural way to characterize the self space is to define a degree of normalcy; this can be accomplished, for instance, by defining the self as a fuzzy set.
- Other immune-inspired algorithms use higher-level representation (e.g., real-valued vectors). A low-level representation, such as binary, makes it difficult to integrate the NS algorithm with other immune algorithms.

This chapter describes a real-valued negative-selection (RNS) algorithm, which uses different encoding schemes to speed up the detector generation process and to alleviate the limitations previously mentioned.

7.3 RNS

The RNS algorithm applies a heuristic process that iteratively changes the position of the detectors. It is driven by two goals: to maximize the coverage of the nonself subspace and to minimize the coverage of the self samples. Different versions of RNS algorithms are being studied for the generation of variably sized and shaped detectors, including spherical, hyper-rectangular, and fuzzy-rule detectors [11,23]. In all these cases, the self/nonself space, U, corresponds to a subset of R^n, unitary hypercube $[0,1]^n$, and each detector covers some nonself area in this high-dimensional space.

7.3.1 Negative Selection for Detection Rules (NSDR)

The first approach uses real-valued representation to characterize the self/nonself space and evolves a set of detectors that can cover the (nonself) complementary subspace (as shown in Figure 7.2). The structure of these detection rules is (R^1, R^2, …, R^m):

$$R^i \geq \text{If Cond}_i X \{= x_1 \in [\text{low}_1^i, \text{high}_1^i] \text{ and } \ldots \text{ and } x_n \in [\text{low}_n^i, \text{high}_n^i]\} \text{ then NonSelf}$$

where $X = (x_1, \ldots, x_n)$ is a feature vector, and $[\text{low}_i^i, \text{high}_i^i]$ specifies the lower and upper values for the feature x_i in the condition part of the rule R^i.

The condition part of each rule defines hyper-rectangle in the self/nonself space, $[0.0,1.0]^n$. A set of these rules tries to cover the nonself space

AU3045_C007.fm Page 174 Saturday, June 25, 2005 9:06 AM

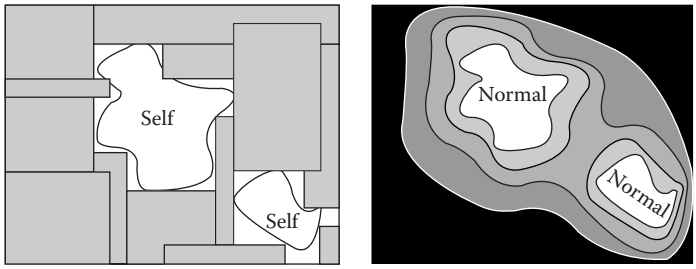


Figure 7.2 Self/nonself space. (a) Approximation of the nonself space by rectangular interval rules. (b) Levels of deviation from the normal in the nonself space.

with hyper-rectangles. For the case $n = 2$, the condition part of a rule represents a rectangle. Figure 7.2(a) illustrates an example of such a coverage 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:

$$X_{\text{non\_self,}}\ R(\vec{x}) = \begin{cases} 1 & \text{if } R^j \quad \text{in } R \text{ such that } x \in R^j \\ 0 & \text{otherwise} \end{cases}$$

Alternatively, the nonself space can be divided into different levels of deviation. In Figure 7.2(b), these levels of deviation are shown as concentric regions around the self regions.

To characterize the different levels of abnormality, we considered a variability parameter (v) to the set of normal descriptor samples, in which v represents the level of variability that we allow in the normal (self) space. A higher value of v means more variability (allows larger variation in self characterization); a lower value of v represents less variability (a smaller self space). Figure 7.3 shows two sets of rules that characterize self subspaces with large and small values of v. Figure 7.3(a) shows coverage using smaller v. Figure 7.3(b) shows coverage using a larger value of v. The variability parameter can be assumed as the radius of a hypersphere around the self samples. Figure 7.3(c) shows the levels of deviation defined by two coverings.

In the nonself space, different values of v are used to generate a set of rules that can provide maximum coverage. An example of such a set of rules:

AU3045_C007.fm Page 175 Saturday, June 25, 2005 9:06 AM

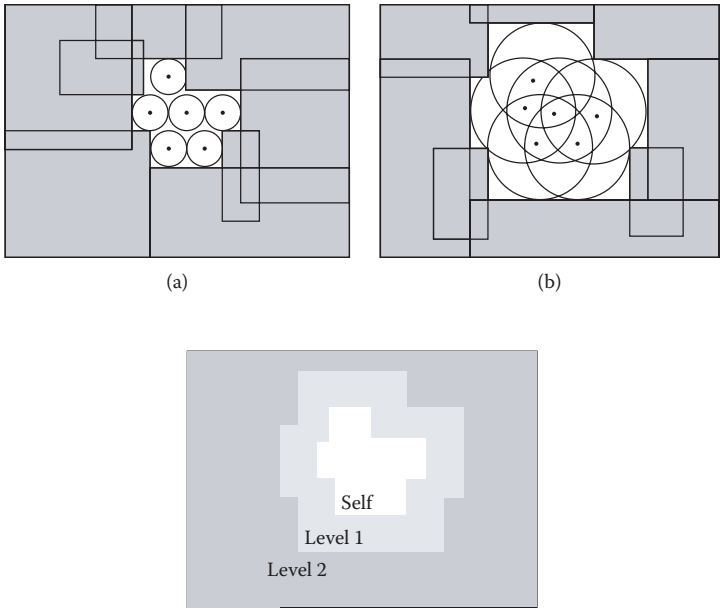


Figure 7.3 A set of normal samples is represented as points in 2-D space. The circle around each sample point represents the allowable deviation. (a) Rectangular rules cover the nonself (abnormal) space using a small value of *v*. (b) Rectangular rules cover the nonself space using a large value of *v*. (c) Level of deviation defined by each *v*, in which level 1 corresponds to nonself cover in (a) and level 2 corresponds to nonself cover in (b).

R^1: If $Cond_1$ then **Level 1**
. . .
. . .
. . .
R^i: If $Cond_i$ then **Level 1**

R^{i+1}: If $Condi_{+1}$ then **Level 2**
. . .
. . .
. . .

R^j: If $Cond_j$ then **Level 2**
. . .
. . .
. . .

AU3045_C007.fm Page 176 Saturday, June 25, 2005 9:06 AM

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 an element in the self/nonself space can be matched by more than one rule, but the highest level reported will be assigned as its level. This set of rules generates a graded characteristic function for the nonself space:

$$\mu_{non\_self}(\vec{x}) = \max(\{l \mid R^j \quad R,\vec{x}\ R^j \text{ and } l = \text{level}(R^j)\} \cup \{0\})$$

where *level* R^j represents the deviation level reported by the rule R^j.

7.4 Intrusion Detection Problem

The anomaly-based intrusion detection problem can be viewed as a learning task that tries to induce, from a training set, a general function that can discriminate between normal and abnormal samples. However, in many anomaly detection problems, only normal samples are available for training. This means that the application of a conventional classification algorithm is not straightforward.

7.4.1 Positive or Negative Characterization?

Normal behavior or normal data patterns will be represented by a subspace S (called SELF) of the feature space, X. On the other hand, the complement of S, $N = X–S$, will be referred to as NON_SELF. The techniques to generate detectors can be classified as follows [19]:

- *Positive characterization (PC):* All the representative patterns are chosen from the set of normal patterns, i.e., normal entities of the system, denoted by S.
- *Negative characterization:* All the representative patterns are chosen from the set of patterns in $X–S$, i.e., abnormal entities of the system.

AU: Sentence incomplete.

The most existing approaches use to build models of the normal set (PC). It is also possible to accomplish the same goal by building models of the abnormal set (negative characterization), as in the NSA (Section 7.3). Negative characterization does not seem to be as natural as PC in cases in which the normal is relatively small. So, what is the justification for negative characterization? Esponda and Forrest [18] provided three main reasons:

AU3045_C007.fm Page 177 Saturday, June 25, 2005 9:06 AM

- There is practical evidence that the negative-detection approach works, because it has been applied with some success to solve practical problems.
- From an information theory point of view, characterizing the normal space is equivalent to characterizing the abnormal space.
- Negative characterization is more suitable for distributed anomaly detection. That is, it is possible to divide a set of negative detectors into subsets and apply them into a distributed fashion, because the activation of only one negative detector is enough to classify a sample as abnormal. If we use positive detection, it is necessary to apply all the positive detectors before it can be concluded that a sample is abnormal.

The third reason appears to be the strongest. However, if the description of the normal set is compact enough, it would be more efficient to have multiple, redundant copies of positive detectors perform distributed anomaly detection. Accordingly, negative detection is more suitable than positive detection for performing distributed detection but only if the normal subspace is not very small.

AU: Words missing?

Keogh et al. [31] argued that "a major limitation of the approach (negative selection) is that is only defined when the space of self is not exhaustive." The authors provided an example of random walk data series, in which the self set can have all possible patterns, causing the nonself set to be an empty set. Notice that this is also a possible issue for the positive detection strategy and, in general, for any learning strategy that tries to induce a model of the normal profile from samples. So, the problem is not associated to the algorithm itself, but to the set of features selected to represent the system behavior, which are not useful to characterize the system or process normalcy. For instance, in the case of the random walk time series, a set of features that includes high-level statistical characteristics of the time series may perform better than a set of features based on a sliding-window scheme.

Depending on the type of datasets to be monitored, different representations may be used for the elements in the parameter (or feature) space, X. If the parameters correspond to categorical data, then a symbolic representation is suitable. Typically, data items are represented as binary strings. On the other hand, if the data items are real-valued, then a real-valued representation may be used. Each element of the feature space is described by a set of features. Thus, it can be represented as a m-tuple $(x_1, \ldots, x_m)$, where x_i is in the unit interval [0,1] when x_i is real-valued, otherwise x_i is in a set $\{0, 1, \ldots, M\}$ whenever you are dealing with categorical data. Typically, for numerical features, the feature space is I^m, the unit m-dimensional hypercube.

AU3045_C007.fm Page 178 Saturday, June 25, 2005 9:06 AM

Once a uniform representation for the parameter space is chosen, a set of patterns that correspond to normal entities is presented to the NSA. Such a set is called the *set of self patterns*, and it is used during the learning process to determine a set of representative elements that will be used to detect novelties in the system. Depending on the context, a representative pattern is called a *detector* or a *classification rule*. We use these terms interchangeably. In next section, we test the proposed approach with network traffic data.

7.4.2 Dataset

We experimented the proposed approaches with network traffic data. The idea is to examine if the system is able to detect some attacks after it is trained with normal traffic patterns. This dataset is a version of the 1999 DARPA intrusion detection evaluation dataset generated and managed by MIT Lincoln Labs [35]. This data represents both normal and abnormal information collected in a test network, in which simulated attacks were performed. The purpose of this data is to test the performance of intrusion detection systems. The datasets contain normal data (not mixed with attacks) obtained over a period of several weeks. This provides enough samples to train the detection system.

The dataset 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 included some attacks. Some of these were network attacks, and the others were inside attacks. Only network attacks were considered for our testing. These attacks are described in Table 7.1, and the attack timeline is shown in Figure 7.4.

Table 7.1 Second-Week Attack 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

AU3045_C007.fm Page 179 Saturday, June 25, 2005 9:06 AM

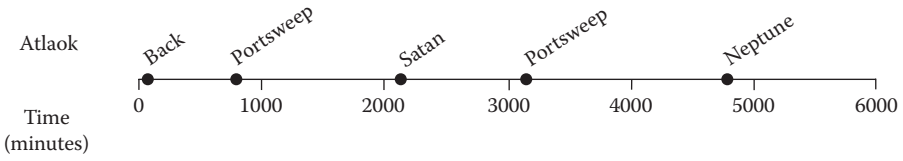


Figure 7.4 Network attacks on the second week.

Table 7.2 Parameters Used and Preprocessing

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
T1	Number of ICMP packets per second	2	Testing

AU: Should this be "T3"?

Three parameters were selected to detect some specific types of attacks. These parameters were sampled each minute (using tcpstat) and normalized. Table 7.2 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:

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

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 to produce the final feature vector. For instance, if we used the three time series S1, S2, and S3 with a window size of 3, a set of 9-dimensional feature vectors was generated.

To evaluate the ability of the proposed approach to produce a good estimation of the level of deviation, we implemented a simple (but inefficient) anomaly detection mechanism. It uses the actual distance of an element to the nearest neighbor in the self set as an estimation of the degree of abnormality.

AU3045_C007.fm Page 180 Saturday, June 25, 2005 9:06 AM

7.5 Experimentation

7.5.1 PC Approach

In this approach, we used the positive samples to build a characterization of the self space, *Self*. In particular, we did not assume a model for the self set. Instead, we used the positive sample set itself 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 nonself set, *non_self*, because its definition is more natural, and the derivation of the self set characteristic function is straightforward.

$$\mu_{non\_self}(\vec{x}) = D(\vec{x}, Self) = \min\left\{d(\vec{x}, \vec{s}) : \vec{s} \in Self\right\}$$

Here, $d(x, s)$ is a Euclidean distance metric (or any Minkowski metric). $D(\vec{x}, Self)$ is the nearest-neighbor distance, that is, the distance from $\vec{x}$ to the closest point in *Self*. Then, the closer an element x is to the self set, the closer the value of $\mu_{non\_self}(x)$ is to 0.

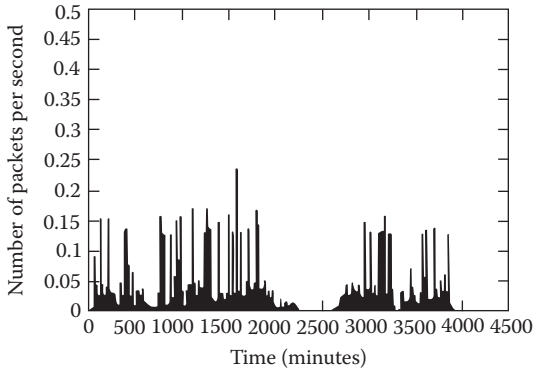
The crisp version of the characteristic function is the following:

$$\mu_{non\_self,t}(\vec{x}) = \begin{cases} 1 & \text{if} \quad \mu_{self}(\vec{x}) > t \\ 0 & \text{if} \quad \mu_{self}(\vec{x}) \le t \end{cases} = \begin{cases} 1 & \text{if} \quad D(\vec{x}, Self) > t \\ 0 & \text{if} \quad D(\vec{x}, Self) \le t \end{cases}$$

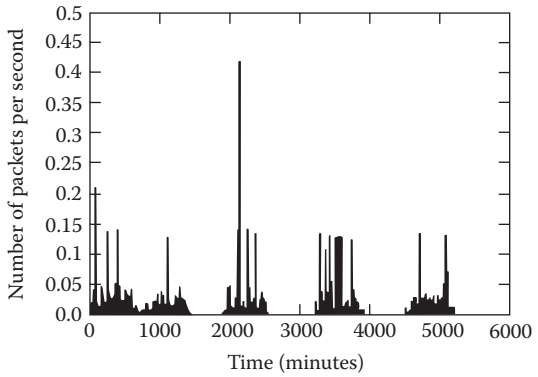
In a dynamic environment, the parameter values that characterize normal system behavior may vary within a certain range over a period of time. The term $(1 - t)$ represents the amount of allowable variability in the self space (the maximum distance that a point can be from the self-samples to be considered normal). This PC can be implemented efficiently by using spatial trees. In our implementation, a KD-tree [5,6] was used. A KD-tree represents a set of k-dimensional points and is a generalization of the standard one-dimensional binary search tree. The nodes of a KD-tree are divided into two classes: internal nodes, which partition the space with a cut plane defined by a value in one of the k dimensions, and external nodes (leaves), which define "buckets" (resulting in hyper-rectangles) in which the points are stored.

This representation allows answering queries in an efficient way. The amortized cost of a nearest-neighbor query is $O(\log N)$ [6]. We used a library (which implements the KD-tree structure) developed at the University of Maryland [37].

AU3045_C007.fm Page 181 Saturday, June 25, 2005 9:06 AM



(a) Training set (S2)



(b) Testing set (T2)

Figure 7.5 Behavior of the parameter *number of packets per second*. (a) Training (self) set corresponding to the first week. (b) Testing set corresponding to the second week.

7.5.1.1 PC Experiments

In each experiment, the training set was used to build a KD-tree to represent the self set. Then, the distance (nearest neighbor) 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; that is, the feature vectors were built using only one variable (time series) each time. Figure 7.5 shows an example of the training and testing datasets for the parameter *number of packets per second*. Figure 7.6(a) represents

AU3045_C007.fm Page 182 Saturday, June 25, 2005 9:06 AM

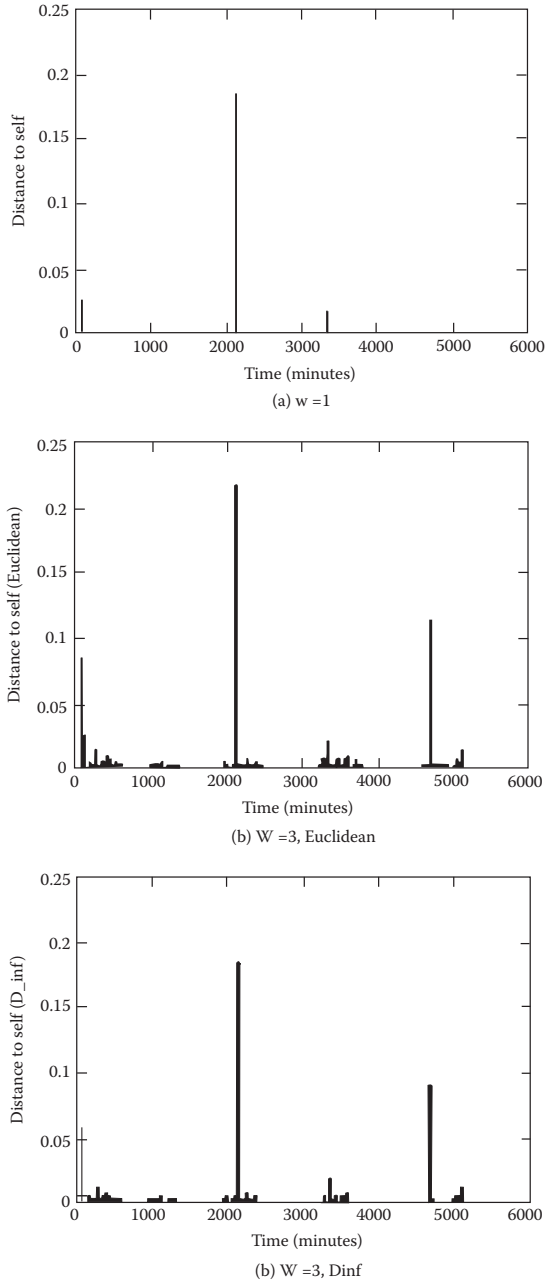


Figure 7.6 Distance from the testing set (T2) to the self set (S2) ($\mu_{non\_self}(x)$). (a) Using window size 1. (b) Using window size 3 and Euclidean distance. (c) Using window size 3 and D_∞ distance.

AU3045_C007.fm Page 183 Saturday, June 25, 2005 9:06 AM

the nonself characteristic function $\mu_{non\_self}(\vec{x})$, that is, 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. Figure 7.6(b) and Figure 7.6(c) show $\mu_{non\_self}(\vec{x})$ for using a window size of 3.

In Figure 7.6(b), the Euclidean distance is used, and in Figure 7.6(c), the D_∞ distance is used.

The plots (in Figure 7.6) 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 (Table 7.1 and Figure 7.6). We conclude the following from these results:

- Using only one parameter is not enough to detect all five attacks. Figure 7.8 shows how the function $\mu_{non\_self}(\vec{x})$ detects deviations that correspond to attacks; however, none of the parameters is able to independently detect all five attacks.
- A higher window size increases the sensitivity; this is reflected in the higher values of deviation.
- 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 in Figure 7.6(a) to 3 in Figure 7.6(b) and Figure 7.6(c), one additional deviation (corresponding 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.
- The change of the distance metric from Euclidean in Figure 7.6(b) to D_∞ in Figure 7.6(c) does not modify the number and type of deviations detected.

As we found in previous experiments, to detect the four attacks, it is necessary to take into account more than one parameter. In the following experiments, we used three parameters to build the feature vector to test whether the PC technique can detect all the attacks. Accordingly, we performed two experiments by varying the sliding-window size.

Figure 7.7 shows the nonself characteristic function for feature vectors conformed to samples of three time series. In all cases, there are five remarkable anomalies that correspond to five attacks. Similar to previous experiments, an increase in the size of the window increases the sensitivity of the anomaly detection function. However, this could generate more

AU3045_C007.fm Page 184 Saturday, June 25, 2005 9:06 AM

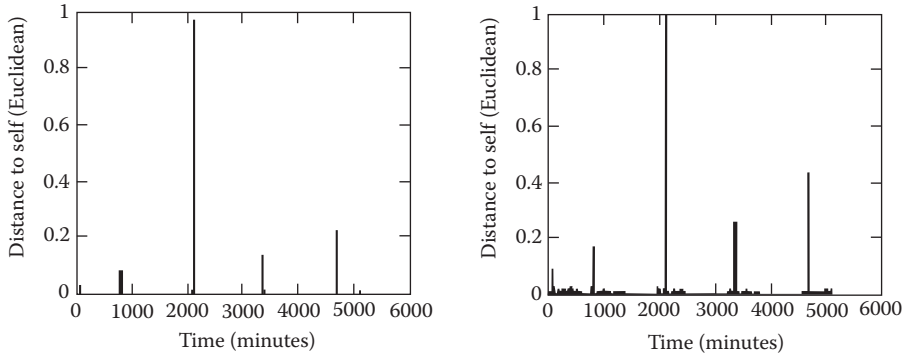


Figure 7.7 Distance from test sets to the self set ($\mu_{non\_self}(\vec{x})$) using S1, S2, and S3. (a) Window size 1. (b) Window size 3.

false-positives. To measure the accuracy of the anomaly detection function, it is necessary to convert them to a crisp version. In this case, the output of the function will be normal or abnormal. This output can be compared with attack information to calculate how many anomalies (caused by an attack) were detected accurately.

The crisp version of the anomaly detection function $\mu_{non\_self}(x)$ is generated by specifying a threshold (t), indicating 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 in the number of detections, but also increasing the number of false-positives (false alarms). To show this trade-off between the false-alarm rate and the detection rate, receiver operating characteristics (ROC) diagrams [39] are drawn. The anomaly detection function μ_{non_self}, $t(x)$ is tested with different values of t, the detection and false-alarm rates are calculated, and this generates a set of points that constitute the ROC diagram. The detection and false-alarm rates are calculated using the following equations:

$$\text{Detection rate} = \frac{TP}{TP + FN} \tag{7.1}$$

$$\text{False-alarm rate} = \frac{FP}{TN + FP}, \tag{7.2}$$

where:

TP: true positives, anomalous elements identified as anomalous
TN: true negatives, normal elements identified as normal

AU3045_C007.fm Page 185 Saturday, June 25, 2005 9:06 AM

FP: false-positives, normal elements identified as anomalous
FN: false-negatives, anomalous elements identified as normal

Figure 7.8 shows the ROC diagrams for the $\mu_{non\_self}(x)$ functions shown in Figure 7.9. In general, the behavior of these 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 performance 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. A possible explanation is that after an attack, some disturbance may still remain in the system, and the function with a larger window size was able to detect it.

The PC technique has been shown to work well on the performed experiments. The main drawback of this technique is its memory requirements, because it is necessary to store the samples that constitute the normal profile. The amount of data generated by network traffic can be large, making this approach unfeasible. This is the main motivation for the negative characterization approach, e.g., NSDR (discussed in the following subsection), compressing the information of the normal profile without significant loss in accuracy.

7.5.2 Evolving Negative-Selection Detection Rules (NSDR)

AU: Change OK?

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. Several criteria guide the evolution process performed by the GA [9,10]. Hence, a rule is considered good if it does not cover positive samples, and its area is large. Accordingly, the soundness of a rule is determined by various factors: the number of normal samples that it covers, its area, and the overlap with other rules. This is a multiobjective, multimodal optimization problem, because a set of rules (solutions) that can collectively solve the problem (covering of the nonself region) is desired.

AU: Check the superscripts.

A niching technique is used with GAs to generate different rules. The input to the GA is a set of feature vectors $S' = \{x^1, \ldots, x^l\}$, which indicate normal behavior. Each element x^j in S' is an n-dimensional vector $x^j = (x_1^j, \ldots, x_n^j)$. The algorithm for the rule generation is shown in Figure 7.9, where:

S':	self-samples training set
v:	level of variability
maxRules:	maximum number of rules in the solution set
minFitness:	minimum fitness 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*

AU3045_C007.fm Page 186 Saturday, June 25, 2005 9:06 AM

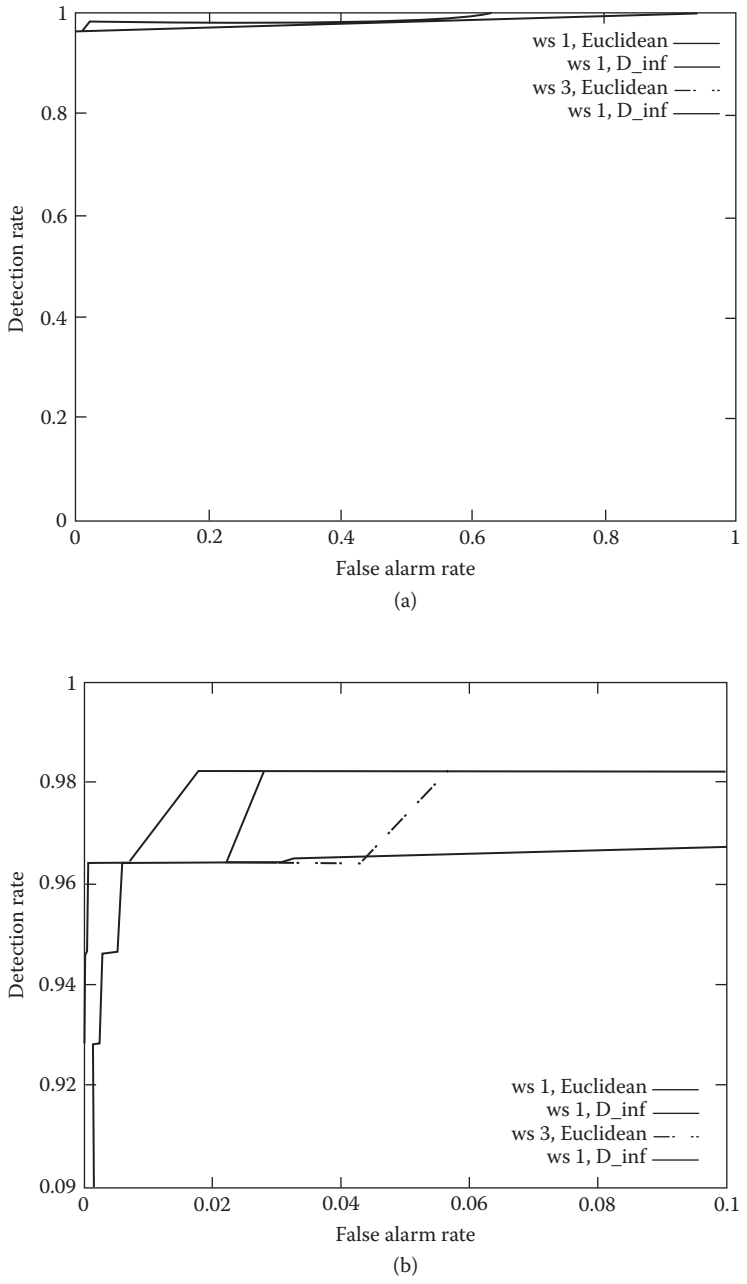


Figure 7.8 ROC diagrams for the $\mu_{non\_self}(x)$ function shown in Figure 7.7. (a) Full scale. (b) Detail of the upper-left corner.

AU3045_C007.fm Page 187 Saturday, June 25, 2005 9:06 AM

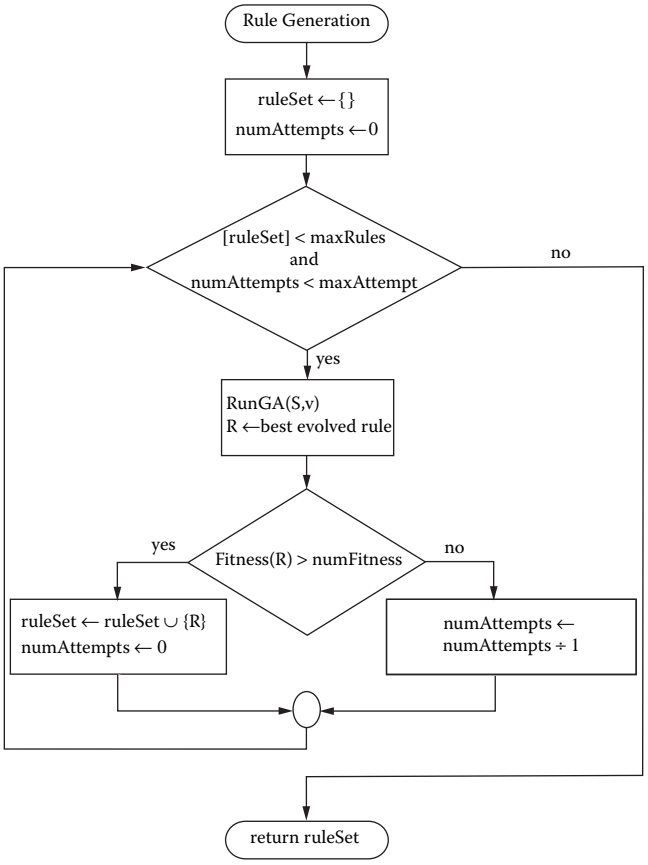


Figure 7.9 NSDR rule generation using a genetic algorithm (GA) with sequential niching (SN).

The algorithm tries to generate a set of rules (*ruleSet*) using a GA (procedure *RunGA*( )). Each rule in *ruleSet* is generated with different runs of the GA. The rule must have a fitness value of at least *minFitness*. If after a maximum number of attempts (*maxAttempts*) it cannot generate a good rule, the algorithm stops (typical values for *maxAttempts* lie between 3 and 5 runs).

The procedure *RunGA*( ) executes a tournament-selection-based GA. Its execution time is O *num_gen pop_size* f_{time}, where *num_gen* is the number of generations, *pop_size* is the population size, and f_{time} is the execution time of the fitness evaluation. In this case, $f_{time} = O(|S|)$, where $|S|$ is the size of the self sample set. Therefore, the execution time of the NSDR algorithm is $O(m.\ num\_gen.\ pop\_size.|S'|)$, where m is the number of generated rules.

AU3045_C007.fm Page 188 Saturday, June 25, 2005 9:06 AM

Each individual (chromosome) in the GA represents the condition part of a rule because the consequent part is the same for all the rules (the descriptor belongs to the nonself). However, the levels of deviation in the nonself space are determined by the variability factor (v). The condition part of the rule is determined by the low and high limits for each dimension. The chromosome that represents these values consists of an array of float numbers. Uniform crossover and Gaussian mutation operators are used.

Given a rule R with a condition part ($x_1 \in [low_1, high_1]$ AND …AND $x_n \in [low_n, high_n]$), 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 hyper-rectangle defined by the points (low_i, …, low_n) and ($high_1$, …, $high_n$).

The raw fitness of a rule is calculated considering the following two factors:

1. The number of elements in the training set S' that are covered by the rule:

$$\text{num\_elements}(R) = \{x^i \in S \mid x^i \in R\}$$

2. The volume of the subspace represented by the rule:

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

The raw fitness is defined as:

$$\text{raw\_fitness}_R = \text{volume}(R) = C.\ \text{num\_elements}(R)$$

where, C is the coefficient of sensitivity. It specifies the amount of penalization that a rule suffers if it covers some normal samples. So, the larger the coefficient (C), the higher is the imposed penalty. Raw fitness can also take negative values.

The idea is to run the GA multiple times [4] to generate different rules so as to cover the entire nonself region. In each run, we want to generate a new rule, that is, a rule that can cover a portion 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.

fitness ← raw_ fitness$_R$
for each $R^j \in$ ruleSet do

AU3045_C007.fm Page 189 Saturday, June 25, 2005 9:06 AM

fitness$_R$ ← raw_ fitness$_R$ = volume(R R^j)
end-For

where *volume*( ) calculates the volume of the subspace specified by the argument.

Because the coverage of the nonself space is accomplished by a set of rules, it is necessary to evolve multiple rules. To evolve different rules, a sequential niching (SN) algorithm is applied.

7.5.2.1 Experiments: NSDR-GA with SN

To test the negative characterization approach (NSDR), we used the MIT DARPA 99 dataset (mentioned in Section 7.4) [35]. We used as training set the time series S1, S2, and S3, and as testing set the time series T1, T2, and T3, with window sizes of 3 and 1, respectively (the time series are described in Table 4.2).

AU: Should this be "Table 7.2"?

The parameters for the GA were 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 values of v). This process was repeated ten times, and the results reported corresponded to the average of these runs.

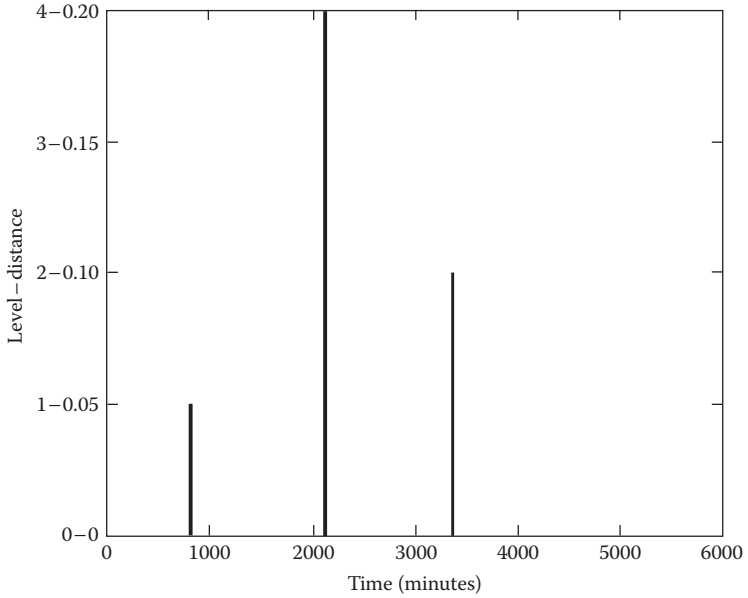
Table 7.3 shows the number of rules generated by the GA for each level. There is a clear difference between the number of rules when the window size changes; the number of rules changes with the size of the window as the pattern space becomes larger.

Figure 7.10 shows two typical attack profiles produced by evolved rules applied to the testing set. With a window size of 1, three out of five attacks are detected, whereas with a window size of 3, four out of five attacks are detected.

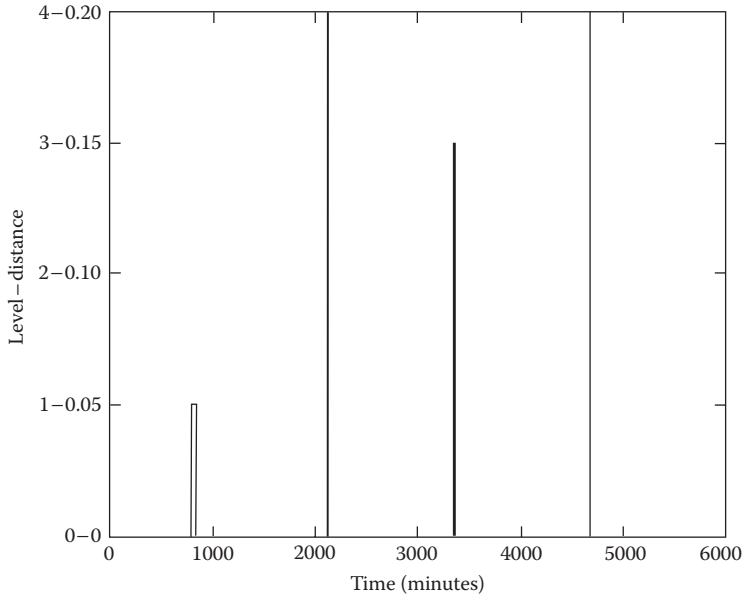
Table 7.3 Number of Generated Rules for Each Deviation Level

Level	*Radius*	*Average Number Rules (Window Size = 1)*	*Average Number 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

AU3045_C007.fm Page 190 Saturday, June 25, 2005 9:06 AM



(a) w = 1



(b) w = 3

Figure 7.10 Indicates the deviations in the testing set detected by the evolved rule set. (a) For window size 1. (b) For window size 3.

AU3045_C007.fm Page 191 Saturday, June 25, 2005 9:06 AM

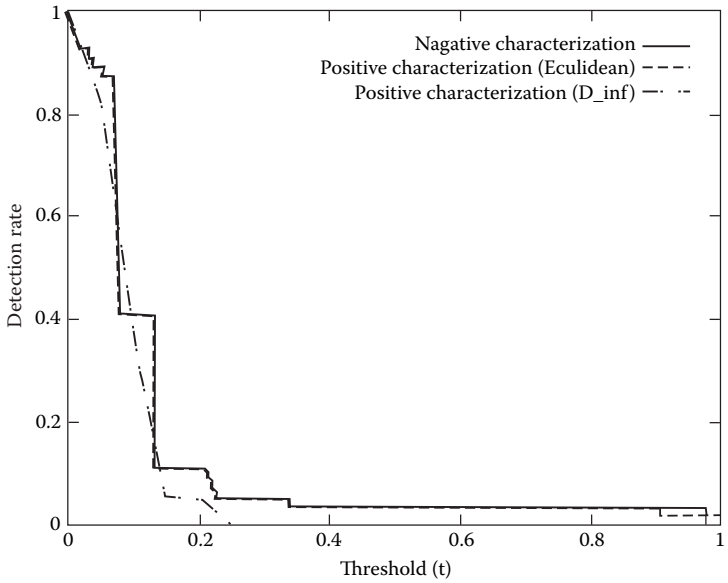
The negative characterization technique (NSDR) is more efficient (in time and space) compared to the PC technique. In the case of a window size of 1, the PC needs to store 5,202 × 3 = 15,606 floating-point values; the NSDR technique only has to store 4 × 6 = 24 floating-point values, so the compression ratio is approximately 1000:1.5. In the case of the window size of 3, the ratio is 46,728:1,698,[3] approximately 100:8. It seems to be a trade-off between compactness of the rule set representation and accuracy. Validity of these arguments is observed in our results. Figure 7.11 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 the NSDR technique, but only by a small difference. In general, the NSDR technique shows detection rates similar to the more accurate (but more expensive) PC technique. Table 7.4 summarizes the best true positive rates (with a maximum false alarm of 1 percent) accomplished by the two techniques. Esponda et al. [19] suggested that this comparison between the PC technique and the NSDR method is not meaningful, because the two methods are quite different. However, the PC technique provides a point reference that facilitates the evaluation of the performance of the NSDR technique.

AU: Check calculation.

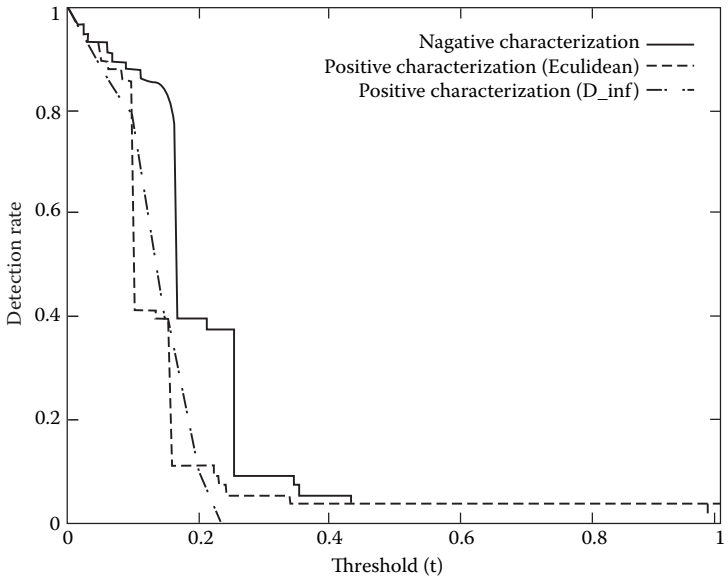
As mentioned earlier, the proposed NSDR technique produces a good estimate of the levels of deviation. To evaluate this estimate, a detailed comparison of the NSDR output levels and PC distance range was performed. The results are illustrated in Table 7.5 in the form of a confusion matrix. For each element in the testing set, the function $\mu_{non\_self}(x)$ generated by the NSDR 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 in square brackets. A perfect output from the NSDR algorithm should generate only values in the diagonal.

The results in Table 7.5 suggest that the NSDR approach better approximates the deviation reported by PC using the D_∞ distance. To support this claim precisely, we measured the number of testing samples for all the possible differences between the PC-reported level and the NSDR-reported level. A difference of zero means that the reported levels are the same, a difference of one means that the results differ by one level, etc. The results for two distances and two window sizes are reported in Table 7.6. The results are very different when different distances are used for the PC algorithm. Clearly, when the D_∞ distance is used in the PC, results of the comparison improved. Despite the fact that only 50.3 percent of the outputs from the NSDR algorithm are same as the PC approach, 100 percent of the NSDR outputs are in the range of 0 or 1 level of difference from that of the PC. The distance metric determines the structure of a metric space. For instance, in a Euclidean space, the set of points that are

AU3045_C007.fm Page 192 Saturday, June 25, 2005 9:06 AM



(a) w = 1



(b) w = 3

Figure 7.11 Comparison of the true positives rate of the detection function μ_{non_self}, $t(x)$ generated by positive characterization (PC) and negative characterization (NSDR) for different values of t. (a) Window size 1. (b) Window size 3.

AU3045_C007.fm Page 193 Saturday, June 25, 2005 9:06 AM

Table 7.4 Best True Positive Rates for the Different Techniques with a Maximum False-Alarm Rate of 1 Percent

Detection Technique	*Window Size 1* (*Percentage*)	*Window Size 3* (*Percentage*)
Positive characterization (Euclidean)	92.8	96.4
Positive characterization (*D*)	92.8	92.8
Negative characterization	82.1	87.5

Table 7.5 Confusion Matrix for PC- and NSDR-Reported Deviations

PC Output Level	*NSDR Output Level*				
Euclidean	*No Deviation* [*0.0,0.05*]	*Level 1* [*0.05,0.1*]	*Level 2* [*0.1,0.15*]	*Level 3* [*0.15,0.2*]	*Level 4* [*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

Note: The values of the matrix elements correspond to the number of testing samples in each class, and the diagonal values represent correct classification.

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 (hyper-rectangle). Therefore, the rectangular rules used by the NSDR approach are better suited to approximate the structure of the D_∞ metric space, and this is reflected in the experimental results.

We investigated GAs to evolve detectors in the complement pattern space to identify any changes in the normal behavior of monitored

AU3045_C007.fm Page 194 Saturday, June 25, 2005 9:06 AM

Table 7.6 The Difference between PC- and NSDR-Reported Levels for Test Dataset

Difference	*Euclidean Distance* (*Percentage*)	D_∞ *Distance* (*Percentage*)
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

Note: The difference is expressed as a percentage of the abnormal feature vectors (distance greater than 0.05). A difference of 0 means that the levels reported by PC and NSDR are the same; a difference of one means that the results differ by 1 level, etc.

behavior patterns. This technique (NSDR) is used to characterize and identify different intrusive activities by monitoring network traffic, and is compared with the other approach (PC). We used a real-world dataset (MIT Lincoln Labs) that has been used by other researchers for testing different approaches. The following are some preliminary observations from these experiments:

- When PC and NSDR approaches are compared, PC appears to be more precise, but it requires more time and space resources. The negative characterization is less precise but requires fewer resources.
- Results demonstrate that the NSDR 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 percent and a maximum false-alarm rate of 1 percent).
- The best results were produced when we used a window size of 3. We observed that a bigger window size makes the system more sensitive to deviations.

7.5.2.2 NSDR-GA Using Deterministic Crowding

The main drawback of SN approach is that the GA must be run multiple times to generate multiple rules. The deterministic crowding (DC) [34] approach allows the generation of multiple rules in a single run. The NSDR algorithm using DC is shown in Figure 7.12. The main inputs to the algorithm are a set of n-dimensional feature vectors $S = \{x^1, \ldots, x^l\}$,

AU3045_C007.fm Page 195 Saturday, June 25, 2005 9:06 AM


```
NS-DETECTOR-RULES(S', num_levels,{ v1,..............,vnumLevels})
S : set of self samples
num_levels : number of deviation levels
{ v1,..............,vnumLevels}: allowed variability for each level
1:for i = 1 to num_levels
2: initialize population with random individuals
3: For j = 1 to num_gen
4: For k = 1 to pop_size/2
5: select two individuals,(parent1 parent2) , with uniform probability
and without replacement
6: apply crossover to generate an offspring(child)
7: mutate child
8: If dist ( child, parent1) < dist( child, parent2)
^ fitness(child)> fitness( parent1)

10: Then parent1 ← child
11: ElseIf dist(child, parent1)>= dist(child, parent2)
12: ^ fitness(child)> fitness(parent2)

13: Then parent2 ← child
14: EndIf
15: EndFor
16: EndFor
17: extract the best individuals from the population and add them to
the final solution
18:EndFor
```

Figure 7.12 Evolving negative-selection detection rules (NSDR) using deterministic crowding (DC).

which represents samples of the normal behavior of the parameter, the number of different levels of deviation (*num_levels*), and the allowed variability for each level $\{v_1, \ldots, v_{numLevels}\}$. Additional parameters to the algorithm are the population size (*pop_size*) and number of generations (*num_gen*).

The execution time of this algorithm is $O(num\_levels.num\_gen.pop\_size\ |S^1|$, where $|S^1|$ is the number of self-samples, which is included in the expression because the time complexity of the fitness calculation is $O(|S^1|)$. Notice that the time complexity depends on the number of levels and not on the number of rules; this makes this algorithm more efficient than the NSDR algorithm based on SN. A good measure of distance between individuals is important for DC niching, because it allows the algorithm to replace individuals with closer individuals. This allows the algorithm to preserve niche formation. The distance measure used in this work is the following:

$$dist(c, p) = \frac{volume(p) - volume(p \cap c)}{volume(p)},$$

where c is a child, and p is its parent.

AU3045_C007.fm Page 196 Saturday, June 25, 2005 9:06 AM

Note that the distance measure is not symmetric. The purpose is to give more importance to the area of the parent that is not covered. The justification is as follows: if the child covers a high proportion of the parent, that means that the child is a good generalization of it, but if the child covers only a small portion, then it is not so. We used the same dataset as before as the training set time series S1, S2 and S3, and as the testing set time series T1, T2 and T3, with a window size of 3. This means that the size of the feature vectors was 9.

The parameters for the GA were population size 200, number of generations 2000, mutation rate 0.1, and coefficient of sensitivity 1.0 (high sensitivity). The GA was run with variability for each level equal to 0.05, 0.1, 0.15, and 0.2, respectively. Then, the elements in the testing set are classified using rules generated for each level (radius). This process is repeated ten times, and the results reported correspond to the average of these runs.

Table 7.7 shows the number of rules NSDR generated by the GA with two niching techniques (NSDR with SN and NSDR with DC). The DC technique produces less rules, which suggests the possibility that the DC technique is discarding some good rules and, therefore, ignoring some niches. However, the performance of the set of rules generated by each technique is apparently similar. This shows that the DC technique is able to find a set of more compact rules producing the same performance. This can be explained by the fact that SN is more sensitive to the definition of the distance between individuals than DC.

Another notable point is the efficiency of the DC technique, as it only needs four runs (one per level) to generate a rule set. For the NC technique, it is necessary to run the GA as many times as the number of rules we want to generate. This is a clear improvement on computational time.

AU: Should this be "SN"?

AU: Subject match not found.

In Section 5.1, it is shown that the NSDR with NC technique produces a good estimate of the level of deviation when this is calculated using the D_∞ distance. Table 7.8 shows the confusion matrix for the NSDR technique using SN and DC. For each element in the testing set, the

AU: Should this be "SN"?

Table 7.7 Number of Generated Rules for Each Deviation Level

Level	*Radius*	*Average Number Rules*	
		Sequential Niching	*Deterministic Crowding*
1	0.05	19.5	7.75
2	0.1	20.7	8.25
3	0.15	26	10
4	0.2	28	10

AU3045_C007.fm Page 197 Saturday, June 25, 2005 9:06 AM

Table 7.8 The Values of the Matrix Elements Correspond to the Number of Testing Samples in Each Class

	NSDR Output Level				
	Sequential Niching				
PC Output Level	0	1	2	3	4
1: [0.0,0.05]	5132	0	0	0	0
2: [0.05,0.1]	3	7.8	0.2	0	0
3: [0.1,0.15]	0	18.1	3.9	0	0
4: [0.15,0.2]	0	0	6.9	9.5	0.6
5: [0.2,...]	0	0	0	1	9
	Deterministic Crowding				
	0	1	2	3	0
1: [0.0,0.05]	5132	0	0	0	0
2: [0.05,0.1]	3	4	4	0	0
3: [0.1,0.15]	0	0	22	0	0
4: [0.15,0.2]	0	0	0	17	0
5: [0.2,...]	0	0	0	0	10

Note: The diagonal values represent correct classification.

function $\mu_{non\_self}(x)$ generated by the NSDR is applied to determine the level of deviation. This level of deviation is compared with the distance range reported by the PC algorithm (using the D_∞ distance). Each row (and column) corresponds to a range or level of deviation. The ranges are specified on square brackets. A perfect output from the NSDR algorithm will generate values only in the diagonal.

In both cases, the values are concentrated around the diagonal, indicating that the two techniques produced a good estimate of the distance to the self set. However, the NSDR approach with DC appears to be more precise. One possible explanation for this performance difference seems to be the fact that the SN requires derating the fitness function for each evolved rule. This arbitrary modification in the fitness landscape can prevent evolving better rules in subsequent runs.

7.5.3 Extending NSDR to Use Fuzzy Rules

We next extended the NSDR algorithm to evolve fuzzy rules instead of crisp rules. That is, given a set of self-samples, the algorithm will generate

AU3045_C007.fm Page 198 Saturday, June 25, 2005 9:06 AM

fuzzy detection rules in the nonself space that can determine if a new sample is normal or abnormal. The use of fuzzy rules appears to further improve the accuracy of the method and produces a measure of deviation from the normal that does not need to partition the nonself space.

The normal and the abnormal behaviors in networked computers are hard to predict, as the boundaries cannot be well defined. Hence, fuzzy logic can provide varying degrees of normalcy in system behavior.

A fuzzy detection rule has the following structure:

$$\textbf{If } x_1 \in T_1 \wedge \ldots x_n \in T_n \textbf{ then } \text{non\_self},$$

where

$(x_1, \ldots, x_n)$: elements of the self/nonself space being evaluated
T_i: fuzzy set
$\wedge$: fuzzy conjunction operator (in this case, min)

The fuzzy set T_i is defined by a combination of basic fuzzy sets (linguistic values).

Given a set of linguistic values $S = \{S_1, \ldots, S_m\}$ and a subset associated with each fuzzy set T_i,

$$T_i = \bigcup_{S_j \in \hat{T}_i} S_j,$$

where $\cup$ corresponds to a fuzzy disjunction operator. We used the addition operator defined as follows:

$$\mu_{A \cup B}(x) = \min\left\{\mu_A(x) + \mu_B(x), 1\right\}$$

An example of fuzzy detection rules in the self/nonself space with dimension $n = 3$ and linguistic values $S = \{L, M, H\}$:

$$\textbf{If } x_1 \in L \wedge x_2 \in (L \cup M) \wedge x_3 \in (M \cup H) \textbf{ then } \text{non\_self},$$

In our experiments, the basic fuzzy sets correspond to a fuzzy division of the real interval [0.0,1.0] using triangular and trapezoidal fuzzy membership functions. Figure 7.13 shows an example of such a division using five basic fuzzy sets representing the linguistic values low, medium-low, medium, medium-high and high.

AU3045_C007.fm Page 199 Saturday, June 25, 2005 9:06 AM

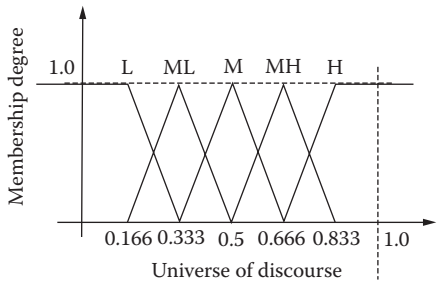


Figure 7.13 Partition of the interval [0,1] in basic fuzzy sets.

Given a set of rules $\{R^1, \ldots, R^k\}$, each one with a condition part $Cond_i$, the degree of abnormality of a sample x is defined by

$$\mu_{\text{non\_self}}(x) = \max_{i=1,\ldots k} \{Cond_i(x)\},$$

where $Cond_i(x)$ represents the fuzzy true value produced by the evaluation of $Cond_i$ in x, and $\mu_{\text{non\_self}}(x)$ represents the degree of membership of x to the nonself set; thus, a value close to 0 means that x is normal, and a value close to 1 indicates that x is abnormal.

To generate the fuzzy-rule detectors, we will use the same evolutionary algorithm described in NSDR with DC. However, the use of fuzzy rules does not require the generation of rules for different levels of deviation. Thus, all the rules are generated in a simple run of the DC algorithm. Figure 7.14 shows the NSFDR algorithm. The time complexity of the algorithm is $O(num\_gen.pop\_size.|Self'|)$.

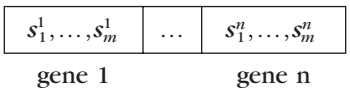
The use of fuzzy rules requires changes in GA implementation such as chromosome representation, fitness evaluation, and distance calculation. These changes are described in the following text.

Each individual (chromosome) in the GA represents the condition part of a rule, because the consequent part is same for all rules (the sample belongs to nonself). As was described before, a condition is a conjunction of atomic conditions. Each atomic condition, x_i, T_i, corresponds to a gene in the chromosome that is represented by a sequence $(s_1^i, \ldots, s_m^i)$ of bits, where $m = |S|$ (the size of the set of linguistic values), and $s_j^i = 1$ if and only if $S_j \subseteq T_i$. That is, the bit s_j^i is "on" if and only if the corresponding basic fuzzy set S_j is part of the composite fuzzy set T_j. Figure 7.14 shows the structure of a chromosome that is $n \times m$ bits long (n is the dimension of the space and m is the number of basic fuzzy sets).

AU3045_C007.fm Page 200 Saturday, June 25, 2005 9:06 AM


```
NS-FUZZY-DETECTOR-RULES(Sel f )
Sel f' : set of self samples
1: initialize population with random individuals
2: For j = 1 to num_gen
3: For k = 1 to pop_size/2
4: select two individuals,(parent1 parent2) ,with uniform probability
and without replacement
5: apply crossover to generate an offspring (child)
6: mutate child
7: If dist( child, parent1)< dist( child, parent2)
^ fitness( child)> fitness( parent1)
8: Then parent1 ← child
9: ElseIf dist( child, parent1)≥ dist( child, parent2)
10: ^ fitness( child)> fitness( parent2)
11: Then parent2 ← child
12: EndIf
13: EndFor
14: EndFor
15: extract the best individuals from the population and add them to
the final solution
```

Figure 7.14 Negative selection with fuzzy detection rules (NSFDR) algorithm.



Given here is the structure of the chromosome representing the condition part of a rule. Each gene represents an atomic condition x_i, T_i, and each bit s_j^i is "on" if and only if the corresponding basic fuzzy set S_j is part of the composite fuzzy set T_j. The fitness of a rule R^i is calculated by taking into account the following two factors:

- The fuzzy true value produced when the condition part of a rule, $Cond_i$, is evaluated for each element x from the self set:

$$selfCovering(R) = \frac{\sum_{x \in Self^t} Cond_i(x)}{|Self^t|}$$

- The fuzzy measure of the volume of the subspace represented by the rule:

$$volume(R) = \prod_{i=1}^{n} measure(T_i),$$

AU3045_C007.fm Page 201 Saturday, June 25, 2005 9:06 AM

where $measure(T_i)$ corresponds to the area under the membership function of the fuzzy set T_i.

The fitness is defined as follows:

$$\text{fitness}(R) = C.(1 - \text{selfCovering}(R)) + (1 - C).\text{volume}(R)$$

where C, $0\ C\ 1$, is a coefficient that determines the amount of penalization that a rule suffers if it covers normal samples. The closer the value of the coefficient to 1, higher is the penalization. In our experiment, we used values between 0.8 and 0.9.

In this work, we used Hamming distance because there is a strong relation between each bit in the chromosome with a single fuzzy set of some particular attribute in the search space. For example, if the s_i^j bit in both parent and child fuzzy-rule detectors is set to 1, both individuals include the atomic sentence $x_i \in s_j$, i.e., they use the jth fuzzy set to cover some part of the ith attribute. Then, the more bits the parent and the child have in common, the more common area they will cover.

7.5.3.1 NSFDR Experimentation

We applied the fuzzy algorithm (negative selection with fuzzy detection rules—NSFDR) and the crisp version (NSDR using DC) to three different datasets as shown in Table 7.9 (two of these are considered here). The algorithms were run 1000 iterations with a population size of 200 individuals. The mutation probability was fixed to 0.1, and the NSDR algorithm was run four times, each time with a different level of deviation (0.1, 0.2, 0.3, and 0.4). The crisp detectors (hyper-rectangles) generated by each run were combined to define the final set of detectors produced by the NSDR.

To access the performance of both methods, we calculate the detection rate (DR, Equation 7.1) and false-alarm rate (FA, Equation 7.2) and plot the result using ROC curves. Also, the reported DR was obtained for each algorithm when the FA was fixed to 3 percent.

Table 7.9 Datasets Used for Experimentation

		Testing	
Dataset	*Training*	*Normal*	*Abnormal*
Mackey-Glass	497	396	101
MIT DARPA 99	4,000	5,136	56
MIT DARPA 98	1,474	19,056	396,745

AU3045_C007.fm Page 202 Saturday, June 25, 2005 9:06 AM

We used the same MIT DARPA 99 dataset described in Section 7.4. Additionally, we used the dataset corresponding to the 1998 version of the DARPA intrusion detection evaluation, also prepared and managed by MIT Lincoln Labs [34]. The dataset was generated by processing the original tcpdump data to extract 42 attributes (33 of them numerical) that characterize the network traffic. This set was used in the *KDD Cup 99* competition and is available at the University of California Machine Learning repository [35]. Even though the dataset corresponds to 10 percent of the original data, its size is still considerably large (492,021 records).

We generated a reduced version of the 10-percent dataset, taking only the numerical attributes. Therefore, the reduced 10-percent dataset is composed of 33 attributes. The attributes were normalized between 0 and 1 using the maximum and minimum values found. Of the normal samples, 80 percent were picked randomly and used as training datasets, whereas the remaining 20 percent was used along with the abnormal samples as a testing set. Five fuzzy sets were defined for the 33 attributes. One percent of the normal dataset (randomly generated) was used as a training dataset (MIT DARPA 98 dataset).

The NSFDR algorithm shows a better performance than the NSDR algorithm (Figure 7.15) with MIT DARPA 98 dataset. The results of the NSDR algorithm are competitive only for a high FA rate (greater than 4 percent). Table 7.10 compares the performance of the tested algorithms and some results reported in the literature. The result produced by the NSFDR algorithm and reported in Table 7.10 is the closest value to the optimum point (0,1). Amazingly, the number of detectors using fuzzyfication is very small compared to the number of detectors using the crisp characterization. This suggests that the fuzzy representation can handle high dimensionality better (the dimensionality of this dataset is 33 attributes).

According to Table 7.10, the performance of NSFDR is comparable with the performance of other approaches reported in the literature and in many cases is better. For example, when NSFDR is compared with RIPPER-AA, the FA rate is almost the same (close to 2 percent), but NSFDR has a higher DR (4 percent more abnormal samples detected). Now, compared with the crisp approach (NSDR), the performance is also superior (2.2 percent more abnormal samples detected). Clearly, the fuzzy characterization of abnormal space reduces the number of false alarms while the detection rate is increased.

When MIT DARPA 99 dataset is used, the performance of the NSDR algorithm is better than that of the NSFDR algorithm for very small values of the FA rate. However, if the FA rate is allowed to be at most 2 percent, the NSFDR is clearly superior (Figure 7.16). Table 7.11 compares the performance of the tested algorithms over the MIT DARPA 99 dataset (for an FA rate less than 3 percent). Again, the fuzzy method (NSFDR) generates

AU3045_C007.fm Page 203 Saturday, June 25, 2005 9:06 AM

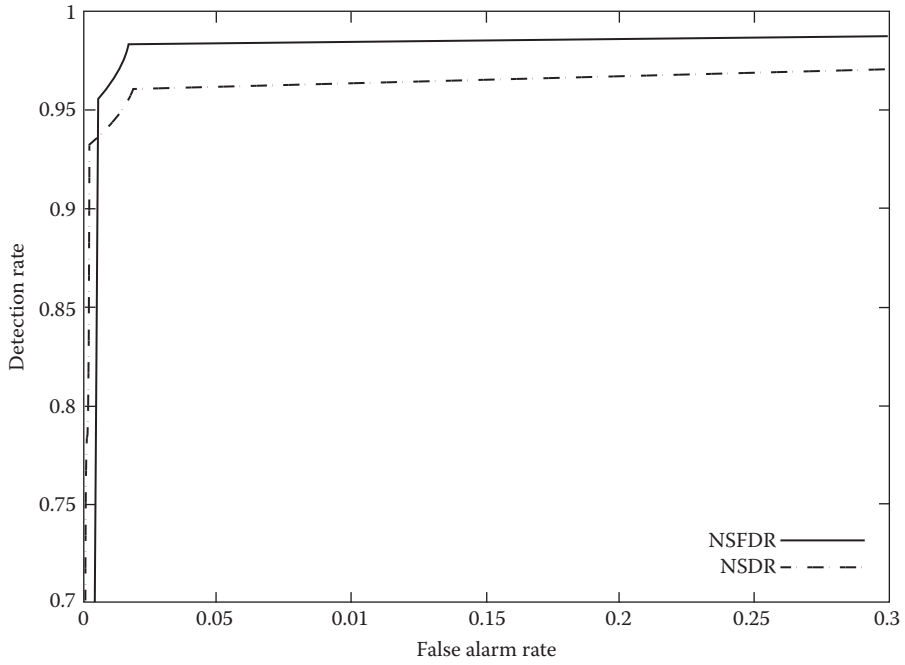


Figure 7.15 ROC curves generated by the two algorithms tested with the MIT DARPA 98 dataset.

Table 7.10 Comparative Performance in the MIT DARPA 98 Dataset

Algorithm	*DR* (*Percentage*)	*FA* (*Percentage*)	*Number of Detectors*
NSFDR	98.22	1.9	14
NSDR	96.02	1.9	699
EFRID[64]	98.95	7.0	—
RIPPER-AA[53]	94.26	2.02	—

a smaller set of rules without sacrificing performance. This supports our claim that the fuzzy representation permits a more compact representation of the self/nonself space.

7.6 Summary

In this chapter, we investigate a technique to perform intrusion detection based on the NSA. Earlier studies showed that binary NS performed well

AU3045_C007.fm Page 204 Saturday, June 25, 2005 9:06 AM

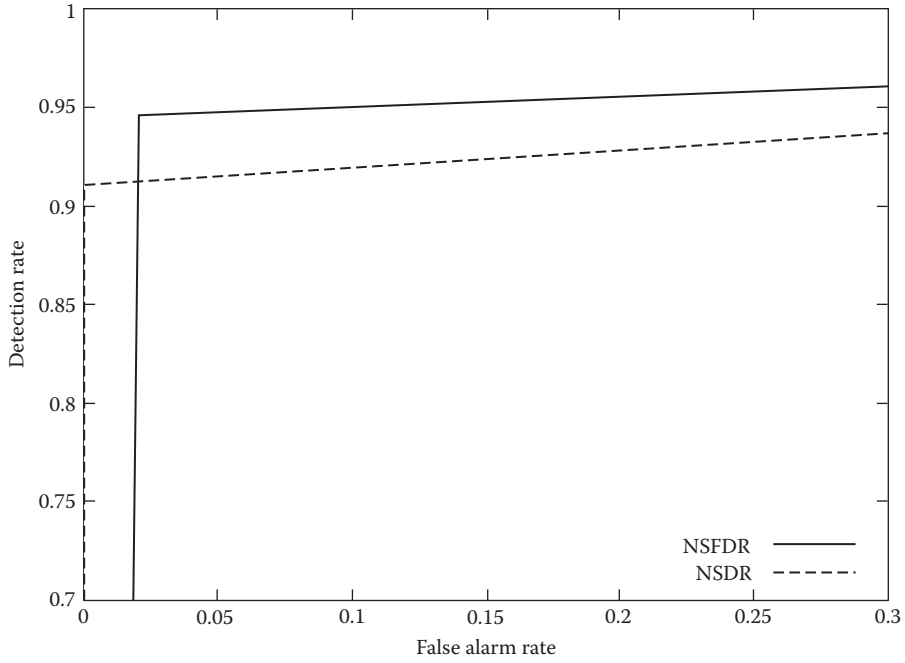


Figure 7.16 ROC curves generated by the two algorithms tested with the MIT DARPA 99 dataset.

Table 7.11 Comparative Performance in the MIT DARPA 99 Problem

Algorithm	*DR* (*Percentage*)	*Number of Detectors*
NSFDR	94.63	7
NSDR	89.37	35

in two of the experiments; however, it failed to produce acceptable results in two other cases. The real-valued NSA takes as input a set of hyperspherical antibodies (detectors) randomly distributed in the self/nonself space. The algorithm applies a heuristic process that changes iteratively the position of the detectors driven by two goals: to maximize the coverage of the nonself subspace and to minimize the coverage of the self-samples. The NSDR algorithm uses a GA to evolve detectors with a hyper-rectangular shape that can cover the nonself space. These detectors can be interpreted as *If-Then* rules, which produce a high-level characterization of the self/nonself space. The first version of the algorithm [11] used an SN technique to evolve multiple detectors. The second version of the

AU3045_C007.fm Page 205 Saturday, June 25, 2005 9:06 AM

algorithm used DC as the niching technique. The algorithm was applied to detect attacks in network traffic data. We further extended the NSDR algorithm to use fuzzy rules, which is called NSFDR. This improves the accuracy of the method and produces a measure of deviation from the normal that does not need a discrete division of the nonself space.

The MIT DARPA 98 dataset is one of the datasets in which BNS failed. This is consistent with the results reported by Kim and Bentley [42,43]; these results were used by them to support the claim that NSA suffers from severe scaling problems. However, our work shows that the problem is not with the NSA itself, but with the kinds of representation (binary) and matching rule (*r*-contiguous) that were used. This was also suggested by Balthrop et al. [3].

The real-valued NSDR technique uses a GA to generate good anomaly detector rules. To test this technique, a set of experiments to detect anomalies in network traffic data was performed. We used a real-world dataset (MIT Lincoln Labs), used by different researchers in computer security, for testing. The following are some preliminary observations:

- The immunogenetic algorithm was able to produce good detectors that gave a good estimation of the amount of deviation from the normal. This shows that it is possible to apply the NSA to detect anomalies on real network traffic data. The real representation of the detectors was very useful in this work.
- The proposed algorithm is efficient; it was able to detect four of the five attacks detected by the PC (with a detection rate of 87.5 percent and a maximum false-alarm rate of 1 percent), while only using a fraction of the space (when compared to PC).
- The use of DC as a niching technique improved the results obtained using SN. While retaining the performance, in terms of a high detection rate, the new algorithm generated a smaller set of rules that estimated the amount of deviation in a more precise way. The new technique is also more efficient in terms of computational power because it is able to generate multiple rules for each individual run of the GA.

AU: Meaning unclear.

The NSFDR technique fuzzy rules as negative detection. The experiments performed showed that the proposed approach performs better than the previous one and is comparable with other results reported in the literature. The following are the main advantages of the NSFDR approach:

- It provides a better definition of the boundary between normal and abnormal. The previous approach used a discrete division of

AU3045_C007.fm Page 206 Saturday, June 25, 2005 9:06 AM

the nonself space, whereas the new approach does not need such a division because the fuzzy character of the rules provide a natural estimate of the amount of deviation from the normal.

- It shows an improved accuracy in the anomaly detection problem. This can be attributed to the fuzzy representation of the rules that reduce the search space, allowing the evolutionary algorithm to find better solutions.
- It generates a more compact representation of the nonself space by reducing the number of detectors. This is also a consequence of the expressiveness of the fuzzy rules.

Bibliography

1. M. Ayara, J. Timmis, L. de Lemos, R. de Castro, and R. Duncan, Negative selection: how to generate detectors, in *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, Canterbury, UK: University of Kent at Canterbury Printing Unit, September 2002, pp. 89–98.
2. J. Balthrop, F. Esponda, S. Forrest, and M. Glickman, Coverage and generalization in an artificial immune system, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, San Francisco, CA: Morgan Kaufmann Publishers, July 9–13, 2002, pp. 3–10.
3. J. Balthrop, S. Forrest, and M.R. Glickman, Revisiting LISYS: Parameters and normal behavior," in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC)*, USA: IEEE Press, 2002, pp. 1045–1050.
4. da. Beasley, D. Bull, and R. Martin, A sequential niche technique for multimodal function optimization, *Evolutionary Computation*, Vol. 1, No. 2, pp. 101–125, 1993.
5. J.L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM*, Vol. 18, No. 9, pp. 509–517, 1975.
6. J.L. Bentley, K-D trees for semidynamic point sets, in *Proceedings of the 6th Annual ACM Symposium Computational Geom*etry, 1990, pp. 187–197.
7. A. Coutinho, The self non-self discrimination and the nature and acquisition of the antibody repertoire, *Annals of Immunology. (Inst. Past.)*, Vol. 131D, 1980.
8. T.M. Cover and P.E. Hart, Nearest neighbor pattern classification, *IEEE Transactions on Information Theory*, Vol. 13, pp. 21–27, 1967.
9. M. Crosbie and E. Spafford, Applying genetic programming to intrusion detection, in *Working Notes for the AAAI Symposium on Genetic Programming*, MIT, Cambridge, MA, USA: AAAI, November 10–12, 1995, pp. 1–8.
10. D. Dasgupta and F. González, Evolving complex fuzzy classifier rules using a linear genetic representation, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, San Francisco, CA: Morgan Kaufmann, July 2001, pp. 299–305.
11. D. Dasgupta and F. Gonzalez, An immunity-based technique to characterize intrusions in computer networks, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 3, pp. 281–291, June 2002.

AU: OK?

AU3045_C007.fm Page 207 Saturday, June 25, 2005 9:06 AM

12. D. Dasgupta, An overview of artificial immune systems and their applications, in *Artificial Immune Systems and Their Applications*, D. Dasgupta, Ed. Springer-Verlag, Berlin, 1999, pp. 3–23.
13. D. Dasgupta and S. Forrest, Tool Breakage Detection in Milling Operations Using a Negative Selection Algorithm, Department of Computer Science, University of New Mexico, Technical Report CS95-5, 1995.
14. D. Dasgupta, *Artificial immune systems and their applications*. Springer-Verlag, Berlin, January 1999.
15. D. Dasgupta, Immunity-based intrusion detection system: a general framework, in *Proceedings of the 22nd National Information Systems Security Conference (NISSC)*, October 1999, pp. 147–160.
16. D. Dasgupta and S. Forrest, Novelty detection in time series data using ideas from immunology, in *Proceedings of the 5th International Conference on Intelligent Systems (ISCA)*, June 1996, pp. 82–87.
17. P. D'haeseleer, S. Forrest, and P. Helman, An immunological approach to change detection: algorithms, analysis and implications, in *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, USA: IEEE Press, 1996, pp. 110–119.
18. F. Esponda and S. Forrest, Detector coverage under the r-contiguous bits matching rule, Department of Computer Science, University of New Mexico, Technical Report TRCS-2002-03, 2002.
19. F. Esponda, S. Forrest, and P. Helman, A formal framework for positive and negative detection schemes, July 2002.
20. S. Forrest and S.A. Hofmeyr, Immunology as information processing, in *Design Principles for the Immune System and Other Distributed Autonomous Systems*, L.A. Segel and I. Cohen, Eds. New York: Oxford University Press, 2000.
21. S. Forrest, A. Perelson, L. Allen, and R. Cherukuri, Self-nonself discrimination in a computer, in *Proceedings IEEE Symposium on Research in Security and Privacy*. Los Alamitos, CA: IEEE Computer Society Press, 1994, pp. 202–212.
22. J. Friedman, J. Bentley, and R. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Transactions on Mathematical Software*, Vol. 3, No. 3, pp. 209–226, 1977.
23. J. Gomez, F. Gonzalez, and D. Dasgupta, An immuno-fuzzy approach to anomaly detection, in *Proceedings of The IEEE International Conference on Fuzzy Systems*, St. Louis, MO, May 2003.
24. S.A. Hofmeyr, An interpretative introduction to the immune system, in *Design Principles for the Immune System and Other Distributed Autonomous Systems*, I. Cohen and L.A. Segel, Eds. New York: Oxford University Press, 2000.
25. S. Hofmeyr and S. Forrest, Architecture for an artificial immune system, *Evolutionary Computation*, Vol. 8, No. 4, pp. 443–473, 2000.
26. C.A. Janeway, How the immune system recognizes invaders, *Scientific American*, Vol. 269, No. 3, pp. 72–79, 1993.
27. C.A. Janeway, P. Travers, S. Hunt, and M. Walport, *Immunobiology: The immune system in Health and Disease*, Garland Pub., New York, 1997.

AU3045_C007.fm Page 208 Saturday, June 25, 2005 9:06 AM

28. N.K. Jerne, Clonal selection in a lymphocyte network, in *Cellular Selection and Regulation in the Immune Response*, Raven Press, New York, 1974, pp. 39–48.
29. N.K. Jerne, Towards a network theory of the immune system, *Ann. Immunol. (Inst. Pasteur)*, Vol. 125C, pp. 373–389, 1974.
30. J. Kappler, N. Roehm, and P. Marrack, T cell tolerance by clonal elimination in the thymus, *Cell*, No. 49, pp. 273–280, 1987.
31. E. Keogh, S. Lonardi, and B. Chiu, Finding surprising patterns in a time series database in linear time and space, in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, USA: ACM Press, 2002, pp. 550–556.
32. T.B. Kepler and A.S. Perelson, Somatic hypermutation in B-cells: an optimal control treatment, *Journal of Theoretical Biology*, Vol. 164, pp. 37–64, 1993.
33. J. Kuby, *Immunology*, 3rd ed., W.H. Freeman and Co., New York, 1997.
34. S.W. Mahfoud, Crowding and preselection revisited, in *Parallel Problem Solving From Nature 2*, Amsterdam: North-Holland, 1992, pp. 27–36.
35. 1999 Darpa intrusion detection evaluation, MIT Lincoln Labs, 1999. [Online]. Available: http://www.ll.mit.edu/IST/ideval/index.html.
36. P.A. Moss, W.M. Rosenberg, and J.I. Bell, The human T-cell receptor in health and disease, *Annu. Rev. Immunol.*, Vol. 10, No. 71, 1993.
37. D. Mount and S. Arya, ANN: a library for approximate nearest neighbor searching, in *2nd Annual CGC Workshop on Computational Geometry*, 1997. [Online]. Available: http://www.cs.umd.edu/mount/ANN.
38. P. Murphy and D. Aha, UCI Repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science, 1992. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.
39. 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*, CA: Morgan Kaufmann, 1998, pp. 445–453.
40. I. Tizzard, The response of B-cells to antigen, in *Immunology: An Introduction*, 2nd ed., Saunders College Publishing, 1988, pp. 199–223.
41. A. Tyrrell, Computer know thy self: a biological way to look at fault tolerance, in *Proceedings of the 2nd Euromicro/IEEE workshop on Dependable Computing Systems*, Milan, 1999, pp. 129–135.
42. J. Kim and P. Bentley, An evaluation of negative selection in an artificial immune system for network intrusion detection, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, San Francisco, CA: Morgan Kaufmann, 2001, pp. 1330–1337.
43. J. Kim and P.J. Bentley, Toward an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection, in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC),* USA: IEEE press, May 2002, pp. 1015–1020.