A Study of Artificial Immune Systems Applied to Anomaly Detection

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Memphis

Fabio González

May 2003

Dedication

This work is dedicated to my wife and my son:

Leydi and Jacobo.

Acknowledgements

To Dr. Dipankar Dasgupta, who introduced me to this field of research, for his support and continuous encouragement.

To my Dissertation Committee members, Dr. Olfa Nasraoui, Dr. Robert Kozma, and Dr. King -Ip Lin, for their valuable comments, helpful suggestions, and constructive criticism.

To Jonatan, Madhavi, Prasad, Senhua, Ravi and all the members of the Intelligent Security Systems Research Lab, for all their help and friendship that made this time much more enjoyable.

To the National University of Colombia, for the financial support to pursue this doctorate.

A very special thanks to all the members of my family, for their unconditional love, support, and encouragement through this process.

Abstract

González, Fabio Ph.D. The University of Memphis. May 2003. A Study of Artificial Immune Systems Applied to Anomaly Detection. Major Professor: Dipankar Dasgupta, Ph.D.

The main goal of this research is to examine and to improve the anomaly detection function of artificial immune systems, specifically the negative selection algorithm and other self/non-self recognition techniques. This research investigates different representation schemes for the negative selection and proposes new detector generation algorithms suitable for such representations. Accordingly, different representations are explored: hyper-rectangles (which can be interpreted as rules), fuzzy rules, and hyper-spheres. Four different detector generation algorithms are proposed: Negative Selection with Detection Rules (NSDR, an evolutionary algorithm to generate hyper-cube detectors), Negative Selection with Fuzzy Detection Rules (NSFDR, an evolutionary algorithm to generate fuzzy-rule detectors), Real-valued Negative Selection (RNS, a heuristic algorithm to generate hyper-spherical detectors), and Randomized Real-valued Negative Selection (RRNS, an algorithm for generating hyper-spherical detectors based on Monte Carlo methods). Also, a hybrid immune learning algorithm, which combines RNS (or RRNS) and classification algorithms is developed. This algorithm allows the application of a supervised learning technique even when samples from only one class (normal) are available. Different experiments are performed with synthetic and real world data from different sources. The experimental results show that the proposed representations along with the proposed algorithms provide some advantages over the binary negative selection algorithm. The most relevant advantages include improved scalability, more expressiveness that allows the extraction of high-level domain knowledge, non-crisp distinction between normal and abnormal, and better performance in anomaly detection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The construction of artificial systems by drawing inspiration from natural systems is not a new idea. Artificial neural networks and evolutionary computation are good examples of successful applications of the biological metaphor to the solution of computational problems. The study of artificial immune systems (**AIS**) is a relatively new field that tries to exploit the mechanisms of the natural immune system (**NIS**) in order to develop problem solving techniques.

The NIS is one of the most complex systems in nature. Its complexity is such that it can be compared with the nervous system [1]. The main purpose of the immune system is to protect the body from damage that can be caused by harmful entities that are mostly foreign, but in some cases, damage can be originated in the body itself [2]. The diversity of these threats is such that they have to be repelled with a comparable variety of overlapping mechanisms. This makes the NIS complex and difficult to understand; in fact, many of these mechanisms are not fully understood [3]. However, this complexity also represents an advantage since it provides a rich source of inspiration for bio-inspired computing.

There are two main parts of the NIS, the innate immune system and the adaptive immune system [2]. When an attack occurs, the innate immune system is the first one to generate a response. This response is not specific, but in many cases, it is able to repel the attack. If the innate immune system fails in thwarting the intrusion,

then the adaptive immune system takes over. The adaptive response is more specific, resulting in a more effective response.

From a computational point of view, adaptive immunity is the most interesting part of the NIS [4]. The adaptive immune system can remember past encounters with antigens (virus, bacteria, foreign molecules, etc) in such a way that the next time the antigen appears, a more specific and effective response is deployed. This mechanism is called *immune memory* [2]. Another interesting mechanism of the adaptive immune system is the *self/non-self recognition* [5]. The NIS is able to recognize which cells are its own (self) and which are foreign (non-self); thus, it is able to build its defense against the attacker instead of self-destructing.

There is a growing area of AIS research [4], but it can be roughly classified in two main categories: techniques inspired by the self/non-self recognition mechanism and those inspired by the immune memory mechanism.

One of the first works that suggested NIS as an inspiration to develop a computational system was performed by Farmer et al. [6]. In this work, he proposed a pattern recognition model based on the idiotypic network theory[7], which explains the immune memory mechanism. This work shows that the NIS can be viewed as a learning system and suggests that it can be used as an inspiration to build machine learning techniques. The work of Cook [8] follows these ideas to develop an algorithm for DNA sequence classification. Timmis [9] modified the algorithm to develop a general technique for data reduction and clustering.

Forrest and her group [10] proposed the *negative selection* (**NS**) algorithm. This algorithm is inspired by the mechanism used by the immune system to train the T-cells to recognize antigens (non-self) and to prevent them from recognizing the body's own cells (self). Different variations of these algorithm were applied for anomaly detection problems [11, 12], for fault detection problems [13, 14], for detection of novelties in time series [15], and even for function optimization [16].

Despite the success of these AIS techniques, there remain many issues unaddressed. As the field is relatively new, most of the existing works have been exploratory, and

the algorithms do not scale. Among others, the following are some aspects that have to be addressed in order to make AIS an useful problem solving technique:

- Improvement of the efficiency of the algorithms.
- Enhancement of the representation.
- Introduction of other mechanisms of NIS that are not used in the current techniques.
- Development of unified models that integrate more natural mechanisms from the AIS.

1.1 Goals

The purpose of this dissertation is to contribute some ideas that will be, hopefully, a step forward in finding the solution to the above mentioned issues. The main goal of this research is to improve the anomaly detection function of AIS, specifically the negative selection algorithm and other self/non-self recognition techniques. The following is a description of the main objectives of the proposed research:

- To study new encoding schemes for the self/non-self representation, instead of the existing low-level binary representation for the NS algorithm.

  The idea is to use representation schemes that are closer to the problem space. This makes the extraction of useful knowledge easier since the NS algorithm deals with objects that have a rich semantic content. Thus, the NS algorithm will be not only a tool to detect anomalies in a given system, but also a tool to produce high-level characterization of the system structure.

- To define different representations of the detectors.

  This is a direct consequence of the previous point. New representations of the self/non-self space requires different detectors and matching schemes. A higher

level representation opens interesting possibilities for defining detectors that have a better semantic content.

- To design new detector generation algorithms.

  The existing NS algorithms are specific to the binary representation; however, a new representation requires new algorithms. These new algorithms might exploit the advantages that the representations provide in order to improve their efficiency and scalability.

- To perform a non-crisp distinction between self and non-self.

  Most of the self/non-self recognition algorithms perform a crisp distinction between the two sets. It is clear that the self/non-self boundary, in general, is not well-defined. Therefore, it is necessary to design detection schemes that can take into account this fuzziness.

1.2 Main contributions

As it was mentioned in the previous section, the main goal of this work is to study alternatives to the binary representation used by the current implementations of the NS algorithm. Our work in studying binary matching rules [17] presents a justification for exploring different alternatives to binary representation. Specifically, this work shows, experimentally, that binary matching rules have a very low-level representation that is unable to capture the structure of even simple problem spaces. The main conclusion is that the matching rule, which determines the representation of the self/non-self space, needs to be chosen in such a way that it represents accurately the affinity relationship in the problem space.

As an alternative to binary representation, we chose a real-valued vector representation of the self/non-self space for the following reasons:

- It is appropriate for multiple applications, even if the data is not real, it is possible to find a mapping to the $\mathbb{R}^n$ space.

- Many AISs use this kind of representation. This implies that the representation is not unnatural and that there is already a conceptual *mapping* from immunology to this representation. For instance, real representation of antibodies and antigens, as well as different matching or affinity functions for this representation, have been extensively studied [18, 19, 20, 9]. Also, the use of this representation allows researchers to integrate the NSA with other immune-based algorithms.

- It is possible to use geometric properties of $\mathbb{R}^n$ to speed up the algorithm. The richer structure of the representation space promotes the application of some heuristics that help to distribute better the detectors on the non-self space, minimizing the overlap and maximizing the coverage.

The multidimensional real-valued representation of the self/non-self space opens the possibility to define different types of detectors. We explored different options and developed detector generation algorithms for each one of them. The following is a list of the algorithms proposed in this dissertation:

- Negative selection with detector rules (**NSDR**)

  This algorithm uses a genetic algorithm to evolve detectors with a hyper-rectangular shape that can cover the non-self space. These detectors can be interpreted as *If-Then* rules, which produce a high-level characterization of the self/non-self space. The initial version of the algorithm [21] used a sequential niching technique to evolve multiple detectors. We developed an improved version of the algorithm [22] using deterministic crowding as the niching technique. The algorithm was applied to detect attacks in network traffic data.

- Negative selection with fuzzy detector rules (**NSFDR**)

  We extended the NSDR algorithm to use fuzzy rules [23]. 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 non-self space.

- Real-valued negative selection (**RNS**)

  This algorithm takes as input a set of hyper-spherical antibodies (detectors) randomly distributed in the self/non-self 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 non-self subspace and to minimize the coverage of the self samples. We combined the algorithm with an hybrid immune learning algorithm [24, 25, 26] and applied it to different data set.

- Randomized Real-valued negative selection (**RRNS**)

  Like the RNS algorithm, the goal of this algorithm is to cover the non-self space with hyper-spherical antibodies. The main difference is that the RRNS algorithm has a good mathematical foundation that solves some of the drawbacks of the RNS algorithm. Specifically, it can produce a good estimate of the optimal number of detectors needed to cover the non-self space, and the maximization of the non-self coverage is done through an optimization algorithm with proved convergence properties. The algorithm is based on a type of randomized algorithms called Monte Carlo methods. Specifically, it uses Monte Carlo integration and simulated annealing.

In many anomaly detection applications, only positive (normal) samples are available for training purpose. However, conventional classification algorithms need both positive and negative samples. We proposed a hybrid immune learning algorithm that combines the RNS algorithm with conventional classification algorithms to perform anomaly detection [24, 26]. This method does not use positive or negative detection; rather, it tries to find a boundary between normal and abnormal classes. Specifically, the hybrid AIS uses normal samples to generate abnormal samples by applying RNS; then, both normal and abnormal samples are used as input to a conventional classification algorithm which produces anomaly classifiers.

1.3 Dissertation outline

Chapter 2 presents a succinct biological background on immunology and discusses the necessary concepts to understand the proposed work. Also, a survey of the area of AIS is given. It is not intended to be comprehensive, but to cover the literature that is directly related to this work. Also, a brief review of different interpretations and approaches to solve the anomaly detection problem is presented. Additionally, a clear statement of the anomaly detection problem, as it is interpreted in the present work, is given.

Chapter 3 shows that binary matching rules with low-level representation are unable to capture the structure of simple problem spaces. In order to support this claim, we use some of the binary matching rules reported in the literature and study how they behave in a bi-dimensional real-valued space. In particular, we study the shape of the areas covered by individual detectors and by a set of detectors generated by the NS algorithm.

Chapter 4 presents the NSDR algorithm and compares it with a positive anomaly characterization approach. The chapter also presents the extension of this algorithm to use fuzzy rules instead of crisp rules (NSFDR). Both algorithms are applied to find anomalies in network traffic data and synthetic time series data.

In Chapter 5, the RNS algorithm is presented. The chapter also presents a new method that combines real-valued negative selection with conventional classification algorithms. The algorithm is tested and compared against an unsupervised learning algorithm using different data sets in anomaly detection.

Chapter 6 describes the RRNS algorithm and its mathematical background. The algorithm is compared experimentally with the RNS algorithm in terms of the coverage of the non-self space. Experiments to validate the assumptions in the design of the RRNS algorithm are performed.

Finally, Chapter 7 presents conclusions, and suggestions for future research.

Chapter 2

Background

2.1 Natural immune system

The immune system 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 to **respond** in order to eliminate or to neutralize the foreign invaders [2].

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 [3]. 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 **non-self** (antigens) [5]. The self/non-self discrimination is an essential characteristic of the immune system, since the outcome of an inappropriate response to self molecules can be fatal.

2.1.1 Structure and function of the immune system

The immune system generates a large variety of cells and molecules for defensive purposes. These cells and molecules together act in a dynamic and intricate network of interactions to detect and eliminate antigens. It is difficult to give a concise picture of such a complex system; moreover, many of the mechanisms are not completely understood. This section gives an abstract view of the immune system, yet omits many details of many specific mechanisms. The purpose is to serve as a reference to the subsequent sections. Detailed review of the the natural immune system and its functionalities may be found elsewhere [3, 27, 2].

The immune system can be envisioned as a multilayer system with defense mechanisms in several layers [28]. The three main layers include the anatomic barrier, the innate immunity and the adaptive immunity. They are described as follows:

2.1.1.1 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 [2].

2.1.1.2 Innate immunity

Innate immunity [3], 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 like 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, since they cannot survive the low pH of the stomach.

Phagocytic barriers Some specialized cells (like 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 that induce the inflammatory response, which is characterized by vasodilation and rise in capillary permeability. These changes allow a large number of circulating immune cells to be recruited to the site of the infection. The cytokines are also produced by other immune cells and non-immune cells, for example those that secrete cytokines when damaged [29].

2.1.1.3 Adaptive immunity

Adaptive immunity [30], also called acquired or specific immunity, represents the part of the immune system that is able to specifically recognize and selectively eliminate foreign microorganism and molecules. The main characteristics of the adaptive immunity [2] are the following:

- **Antigenic specificity.** It allows the immune system to distinguish subtle differences among antigens.
- **Diversity.** The adaptive immune system can generate billions of different recognition molecules that are able to uniquely recognize different structures of foreign antigens.

- **Immunologic memory.** The adaptive immune system can *remember* a previous encounter with an antigen. This helps to deliver a quick response in subsequent encounters.

- **Self/non-self recognition.** As it was mentioned before, the immune system can distinguish its own cells from foreign antigens, and so responds only to the non-self molecules.

It is important to note that the acquired immunity does not act independently of the innate immunity; on the contrary, they work together to eliminate foreign invaders. For instance, the phagocytic cells (innate immunity) are involved in the activation of the adaptive immune response. Also, some soluble factors, produced during a specific immune response, have been found to augment the activity of these phagocytic cells [2].

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. The cells are produced by the bone marrow and are inert initially, i.e. they are not capable of executing their functions. In order 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 which circulate in the body, constituting an effective, distributed anomaly detection and response system [2] .

Humoral immunity Mature B-cells express unique antigen-binding receptors (ABR) on their surface. The interaction of the ABR with specific antigen induces

proliferation and differentiation of B-cells into **antibody**-secreting plasma cells. An antibody is a molecule that binds to antigens and neutralize them or facilitate their elimination. Antigens coated with antibodies can be eliminated in multiple ways: by phagocytic cells, by the complement system, or by preventing them form performing any damaging function (e.g. binding of viral particles to host cells) [31].

Cellular Immunity During their maturation, T-cells express an 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[1], 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 [32] .

Self/non-self discrimination As it 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 non-self peptides presented by MHC. This process has two main phases: positive selection and negative selection [5].

Positive selection During the positive selection phase, T-cells are tested for recognition of MHC molecules expressed on the cortical epithelial cells. If a T-cell fails to recognize any of the MHC molecules, it is discarded; otherwise, it is kept.

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

[1]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 (APC).

test. This process can be seen as a filtering of a big diversity of T-cells; only those T-cells that do not recognize self peptides are kept [33].

Immune memory Immune-competent lymphocytes are able to recognize specific antigens through their ABR . The specificity of each T-cell and each 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 [34]. The presence of an antigen in the system and its subsequent interaction with mature lymphocytes triggers an immune response, resulting in the proliferation of lymphocytes with a unique antigenic specificity. This process of population expansion of particular T-cells and B-cells is called **clonal selection** [35].

Clonal selection contributes to the specificity of the adaptive immunity response since 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 the immune memory [2]. 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 of these lymphocytes 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.

2.1.2 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 [4, 36]:

- **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 the molecular shape and on the electrostatic charge.

- **Feature extraction**: in general, immune receptors do not bind to the complete antigen, rather portions of it (peptides). In this way, the immune system can recognize an antigen just by matching segments of it. Peptides are 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 the interaction with the environment. The first time an antigen is detected, a primary response is induced and includes the proliferation of lymphocytes and 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.

- **Diversity**: clonal selection and hyper-mutation mechanisms are constantly testing different detector configuration for known and unknown antigens. This is a highly combinatorial process that explores the space of possible configurations looking 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 around 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, 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 in order to stop the delivery of new resources and to release the used ones.

- **Self-protection**: by protecting the body as a whole, the immune system is protecting itself. It means that there is no other additional system to protect the immune system; the immune system can self-defend.

2.2 Artificial immune systems

The study and design of artificial immune systems (**AIS**) is a relatively new area of research that tries to build computational systems that are inspired by the natural immune system (**NIS**) [37]. As we mentioned in Subsection 2.1.1.3, there are many desirable computational features in the NIS that can be used to solve computational problems. A typical AIS implements one or more of these features.

In many respects, AISs are abstract computational modeling of the immune system; in fact, some AIS techniques are based on theoretical models of the NIS. 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 [7]. This theory proposed that the NIS 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-cells epitopes. This recognition usually extends to more than one level, resulting in the formation of complex reaction networks.

One of the early works that suggested NIS-based computational algorithms was developed by Farmer et al. [6]. In this work, they proposed a computational model

of the NIS based on the idiotypic network theory of Jerne [7]. This model is a simplification of the NIS that ignores important elements like T-lymphocytes and macrophages and concentrates on the modeling of the idiotypic networks. The model represents antibodies and antigens as sequences of **0**'s and **1**'s and uses different equations to define the dynamic of the system. The authors suggest that the model "*has a strong similarity to an approach to learning and artificial intelligence introduced by Holland, called the classifier system*"[2][6]; this similarity gives foundation to suggest that the NIS can be a good inspiration to build learning systems.

The work of Varela [1] took even further the idiotypic network theory and proposed that this network can be thought of as having *cognitive* capabilities that makes it similar to a neural network. Bersini et al. [38, 39] also presented an immune recruitment mechanism and proposed its application in control engineering.

The last decade has seen an increase in AIS research with a wide variety of works in different areas. Based on the survey of existing AIS literature, we tried to put them in a tabular form. Tables 2.1, 2.2, 2.3, and 2.4 show a chronological list of some AIS models and techniques that we considered more relevant. The tables include a short description of each model or technique, along with the information about immunological mechanisms used, the type of representation, and the intended applications.

The different models/techniques use a variety of NIS aspects; however, the most relevant are the antigen-antibody (**Ag-Ab**) binding[3] (see Subsection 2.1.1.3 on page 12), idiotypic immune network theory (described above), and the self/non-self discrimination (see Subsection 2.1.1.3 on page 12). The modeling of the Ag-Ab biding mechanism is present in almost all the models and techniques. In fact, this feature is the only one that can uniformly characterize most AIS models and distinguish them from other soft-computing models. The representation of the basic elements of the NIS (usually antigens and antibodies) also varies from model to model. Binary and real vectors are the most common representations among different approaches. There are

[2]The classifier system is a genetic-based machine learning system that is composed of syntactically simple classifier rules and is able to interact and learn in a dynamic environment.

[3]In the case of idiotypic immune networks, it also includes antibody-antibody binding.

different areas of application with special emphasis in computer security, anomaly detection, and learning (pattern recognition, data analysis, etc.).

It is difficult to classify the different models and techniques since there is no single criteria that can be derived to build a satisfactory taxonomy. It is possible, however, to distinguish some areas of research that share common elements. In order to organize our discussion, we will divide the different works into the following categories:

- Algorithms based on the self/non-self discrimination mechanism (negative selection).
- Algorithms based on the idiotypic immune network theory of Jerne and the clonal selection mechanism.
- Software and hardware architectures inspired by the organizational structure of the NIS.
- Hybrid models or techniques that combine immune system ideas with other soft computing models.

This broad division of the AIS field does not intend to be an exhaustive and non-intersecting partition of the area. In fact, there are many models that cannot be classified in any of these categories, and others that share elements from more than one. Taking these constrains into account, we will give a general overview of each of these areas in the following subsections.

2.2.1 Negative selection based algorithms

As it was mentioned in Section 2.1.1.3, during the generation of T-cells, receptors are made through a pseudo-random genetic rearrangement process [15, 44, 58]. Then, they undergo a censoring process in the thymus, called the negative selection. There, T-cells that react against self-proteins are destroyed; thus, only those that do not bind to self-proteins are allowed to leave the thymus. These matured T-cells then

Table 2.1: A time-line of AIS (1986 to 1995)

Author (year)	**Model or technique description**	**Aspects of the NIS modeled**	**Type of representation used**	**Applications**
Farmer et al. (1986) [6]	The immune system as machine learning process.	Ag-Ab binding. Immune network.	Binary strings.	NIS modeling.
Bersini and Varela (1991) [40]	Selective evolutionary strategy based on immune recruitment.	Immune network. Recruitment mechanism.	Real-valued vectors.	Optimization.
Forrest et al. (1993) [41]	Exploration of pattern recognition in NIS using genetic algorithms.	Ag-Ab binding.	Binary strings.	NIS modeling.
Forrest et al. (1994) [10]	Original Negative Selection algorithm based on the T-cell recruitment process performed by the thymus.	Ag-Ab binding. Self/non-self discrimination.	Strings from a finite alphabet.	Change and anomaly detection.
Kephart (1994) [42]	A computer immune system architecture to detect and repeal virus.	Ag-Ab binding. Self/non-self discrimination.	Byte strings (signatures).	Computer security.
Ishiguro et al. (1995) [43]	A decentralized behavior arbitration mechanism to control robots inspired by the NIS.	Ag-Ab binding. Immune network. Distributed control.	High level representation (robot instructions).	Robot control.
Hunt and Cooke (1995) [8]	An AIS based on immune network theory for learning.	Ag-Ab binding. Immune network.	Binary strings.	DNA sequence matching. Case based reasoning.

Table 2.2: A time-line of AIS (1996 to 1999).

Author (year)	**Model or technique description**	**Aspects of the NIS modeled**	**Type of representation used**	**Applications**
D'Haeseleer et al. (1996) [44]	An efficient implementation of the negative selection algorithm for binary strings.	Ag-Ab binding. Self/non-self discrimination.	Binary strings.	Change and anomaly detection.
Dasgupta and Forrest (1996) [15]	A method to detect novelties in time series based on the negative selection algorithm.	Ag-Ab binding. Self/non-self discrimination.	Binary string representing real values.	Anomaly and novelty detection.
Ishida (1996) [45]	An agent architecture based on immune networks.	Ag-Ab binding. Immune network.	Real-valued vectors.	Fault diagnosis.
Hajela et al. (1997) [46]	Uses immune networks to improve the convergence of genetic algorithms applied to design optimization.	Ag-Ab binding. Immune network.	Binary strings	Evolutionary design optimization.
Hunt et al. (1999) [47]	A machine learning system (**Jisys**) based on immune networks.	Ag-Ab binding. Immune network.	Mixed numerical, categorical and string data.	Fraud detection. Learning.
Dasgupta (1999) [48]	An architecture for an agent-based intrusion/anomaly detection and response system.	Distributed control. Self/non-self discrimination.	Not apply.	Computer security.
Dasgupta and Cao (1999) [49]	Combines immune system ideas and genetic algorithms to interpret chemical spectra.	Ag-Ab binding. Self/non-self discrimination.	Binary strings.	Chemical spectrum recognition.
Williams et al. (1999) [50]	A multi-agent computational immune system (**CDIS**) for intrusion detection.	Ag-Ab binding. Self/non-self discrimination.	Strings from a finite alphabet.	Computer security.

Table 2.3: A time-line of AIS (2000).

Author (year)	**Model or technique description**	**Aspects of the NIS modeled**	**Type of representation used**	**Applications**
Tarakanov and Dasgupta (2000) [51]	A formal model of the immune system.	Ag-Ab binding.	Real-valued vectors.	NIS modeling.
Timmis (2000) [9]	A resource limited artificial immune system (**RAINE**) for data analysis that extends the work of Cooke and Hunt [8].	Ag-Ab binding. Immune network.	Real-valued vectors.	Data analysis. Clustering.
De Castro and Von Zuben (2000) [19]	A system based on clonal selection and affinity maturation (**CLONALG**) for pattern matching and optimization.	Ag-Ab binding. Clonal selection. Affinity maturation.	Binary and integer strings.	Pattern matching. Optimization.
De Castro and Von Zuben (2000) [18]	An immune network learning algorithm (**aiNet**).	Ag-Ab binding. Clonal selection. Affinity maturation. Immune network.	Real-valued vectors.	Data analysis. Clustering.
Hofmeyr et al. (2000) [12]	An architecture for an artificial immune system (**Lisys**) for computer security.	Ag-Ab binding. Self/non-self discrimination. Affinity maturation.	Binary strings.	Computer security.
Bradley and Tyrrel (2000) [52]	A machine fault tolerance mechanism based on immune system ideas (**Immunotronics**).	Self/non-self discrimination.	Binary strings.	Hardware fault detection and tolerance.

Table 2.4: A time-line of AIS (2001 to 2003).

Author (year)	**Model or technique description**	**Aspects of the NIS modeled**	**Type of representation used**	**Applications**
De Castro and Von Zuben (2001) [53]	A simulated annealing algorithm based on a immune systems (**SAND**) applied to neural network initialization.	Ag-Ab binding. Immune diversity.	Real-valued vectors.	Initialization of feed-forward neural network weights.
Tarakanov and Dasgupta (2002) [54]	An architecture to build chips that implement the immune system model proposed in [51].	Ag-Ab binding. Immune network.	Real-valued vectors (internally represented as bits).	Pattern matching.
Nasraoui et al. (2002) [20]	An immune network based algorithm that uses fuzzy theory to model the Ag-Ab matching.	Ag-Ab binding. Immune network.	Real-valued vectors.	Clustering. Web data mining
Hart and Ross (2002) [55]	A system to cluster non-stationary data (**SOSDM**) that combines ideas from NIS and sparse distributed memories.	Ag-Ab binding. Immune memory.	Binary strings.	Associative memory. Clustering.
Coello and Cortez (2002) [16]	An approach to handle constraints in GA based optimization.	Ag-Ab binding. Gene libraries.	Binary strings	Optimization.
Kim and Bentley (2002) [56]	An algorithm to perform dynamic learning on changing environments .	Ag-Ab binding. Clonal selection. Self/non-self discrimination.	Binary strings.	Dynamic learning.
Nasraoui et al. (2003) [20, 57]	A scalable artificial immune system model for dynamic unsupervised learning based on immune network theory.	Ag-Ab binding. Immune network.	Real-valued vectors.	Clustering. Dynamic learning.

circulate throughout the body to perform immunological functions and protect the body against foreign antigens.

Forrest et al. [10] developed a negative selection (NS) algorithm based on the principles of self/non-self discrimination in the NIS. This negative selection algorithm can be summarized as follows (adapted from [4]):

- 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 as normal for the system.

- Generate a set R of *detectors*, each of which fails to match any string in S. An approach that mimics what happens in the NIS would generate random detectors and discard those that match any element in the self set. However, a more efficient approach will try to minimize the number of generated detectors while maximizing the covering of the non-self space.

- 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 not to match any of the original strings in S.

The previous description is very general and does not say anything about what kind of feature space is used or what *matching* exactly means. It is clear that the algorithmic problem of generating good detectors can be very different depending on the kind of feature space (continuous, discrete, mixed, etc.), detector representation scheme, and the rule that determines if a detector matches an element or not.

The first version of the algorithm was proposed by Forrest et al. [10]. The feature space was restricted to binary strings of fixed length and the matching between detectors and elements was defined by a process called r-contiguous matching. The

matching process was defined as follows: given a binary string $x = x_1x_2...x_n$ and a detector $d = d_1d_2...d_n$,

$$x \text{ matches } d \equiv \exists i \leq n - r + 1 \text{ such that } x_j = d_j \text{ for } j = i, ..., i + r - 1,$$

that is, two strings match if there is a window of size r where all the bits are identical. The algorithm works in a generate-and-test fashion, i.e. random detectors are generated and then are tested for self-matching. If a detector fails to match all self strings, it is retained. The number of random detectors that is required to be generated is exponential on the size of self [59]; this makes the algorithm very inefficient.

Two new detector generation algorithms (based on dynamic programming) were proposed by D'haeseleer et al.[44], the *linear* NS algorithm and the *greedy* NS algorithm. Similar to the previous algorithm, they are also specific to binary string representation and r-contiguous matching. Both algorithms run in linear time and space with respect to the size of the self set, though the time and space is exponential on the size of the matching window, r.

Some alternatives to r-contiguous matching have been proposed [60, 61, 62]. For example, different matching rules (similarity measures) were reported by Harmer et al. [62]; however, an efficient negative selection algorithm that uses them was not presented. An algorithm that extends the exhaustive algorithm was proposed by Castro and Timmis [63]; this algorithm was compared with other implementations by Ayara et al. [60]. Balthrop et al. [61] proposed a new matching rule that subsumes r-contiguous matching, called r-chunks. Some preliminary experiments on a "small data set" suggests that the r-chunk matching rule can improve the accuracy and performance of the NS algorithm.

2.2.2 Idiotypic network and clonal selection based algorithms

Based on the immune network theory [7], Cooke and Hunt [8] proposed a supervised machine learning algorithm to classify DNA sequences as either promoter or

non-promoter classes. The system consisted of a network of B-cells that generated antibodies to classify DNA sequences (corresponding to antigens). Each B-cell can be stimulated (by antigens or by other B-cells) or suppressed (by other B-cells).

Timmis [9] proposed a new algorithm similar to the above, but domain independent, called AINE (Artificial Immune NEtwork). The elements of a training data set correspond to antigens that stimulate a set of B-cells. Each B-cell represents a data element, and the strength of the stimulation is determined by the Euclidean distance between the antigen and the B-cell. As in the previous model, B-cells can be suppressed or stimulated by other B-cells. According to its stimulation level, a B-cell can produce a number of clones that are subsequently mutated. The final outcome of this algorithm is a network of B-cells that follows the structure of the training data. This network constitutes a reduced version of the original data that can be used for data clustering or compression.

One major drawback of AINE is the explosion in B-cell population. This problem is addressed, in part, in another algorithm called RAIN (Resource limited Artificial Immune Network) [64]. The main difference between AINE and RAIN is that the basic element of the RAIN algorithm is not the B-cell but the Artificial Recognition Ball (ARB). Each ARB corresponds to a set of identical B-cells but still represents a single data item. Each ARB is assigned a number of resources (B-cells) depending on its stimulation; these resources, unlike the previous model, are restricted.

Nasraoui et al. [65, 20] proposed an immune network based algorithm (called FuzzyAIS) that uses a fuzzy set to model the area of influence of each B-cell. This improves the expressiveness of previous models and makes it more robust to noise and outliers. The algorithm was applied to mine user profiles in web access data [66].

De Castro et al. [19] proposed an algorithm based on the clonal selection and affinity maturation principles of the NIS called **CLONALG**. The main applications of this algorithm are pattern matching and optimization. It is a population-based, evolutionary-like algorithm guided by mechanisms of reproduction, genetic variation

(mutation only), and selection. So, it has many elements in common with conventional evolutionary algorithms.

A model called **aiNet**, which shares some characteristics of Timmis' AINE, was proposed by De Castro [18]. The main difference is that the immune network structure is not a part of the antibody (B-cell in AINE) cloning and selection process. The final network is extracted from the output data by applying a hierarchical clustering algorithm.

2.2.3 Software and hardware architectures inspired by the organization of the NIS

The models in this category are more general in the sense that they do not belong to any specific algorithm; rather, they are part of a general software architecture inspired by the distributed processing mechanism of the NIS.

Kephart [42] proposed an architecture inspired in the immune systems to protect computer systems from viruses. The basic idea was to extend a conventional signature-based antivirus by adding the possibility of dealing with an unknown virus (virus without a signature). If an unknown virus is detected, the system automatically extracts the signature and adds it to the known-signature data base.

The distributed control of the NIS has served as inspiration to design agent architectures that are able to accomplish tasks without centralized control. Ishida [45] proposed an agent-based immune algorithm to perform fault diagnosis. The most relevant characteristic of the system is the self-adaptivity to changing environment, which could include a change on the self. A decentralized behavior arbitration mechanism for controlling a set of independent robots was proposed by Ishiguro et al. [43]. This mechanism allows a set of independent robots to accomplish cooperative tasks without having a centralized control.

NIS has also been an inspiration for designing network security systems. The main goal of this type of system is to detect anomalies and/or intrusions in net-

worked computers and to deploy responses that prevent a further degradation of the system. Examples of this type of architectures are an architecture for agent-based intrusion/anomaly detection and response system proposed by Dasgupta [48], a multi-agent computational immune system for intrusion detection (**CDIS**) developed by Williams et al. [62, 50], and an architecture for a computer immune system (**Lisys**) designed by Hofmeyr et al.[12].

Another promising area of work is the implementation of AIS in hardware design. Bradley et al. [52, 67] developed a machine fault tolerance mechanism based on the self/non-self discrimination mechanism of the NIS that can be implemented directly in hardware. Tarakanov and Dasgupta [54] proposed a hardware implementation of a formal immune model [51]. This work is the first to explore the feasibility of implementing an AIS in hardware - *immunochip*.

2.2.4 Hybrid immune system models

Researchers also explore combining AIS with other computational models and techniques, especially with soft-computing methods. An interesting characteristic of the NIS is that it combines a diverse repertory of mechanisms that interact in a tight and complex way to accomplish the goal of keeping the body free of foreign attackers. Therefore, it is quite natural to develop AISs that combine different problem solving strategies.

Evolutionary computation shares many elements, concepts like population, genotype-phenotype mapping, and proliferation of the most fitted are present in different AIS methods. Some of the earlier work that combined NIS ideas with evolutionary computation was developed by Bersini and Varela [40]. This work proposed a selective evolutionary technique based on the immune recruitment mechanism. A later work by Hajela et al. [46] used immune networks to improve the convergence of genetic algorithms for design optimization. Dasgupta and Cao [49] developed an immunogenetic technique for chemical spectrum recognition. A more recent work from Coello and

Cruz-Cortés uses an immune inspired approach to handle constrains in genetic based optimization [16] and to solve multi-objective optimization problems using genetic algorithms [68].

AIS models based on immune networks resembles the structures and interactions of connectionist models. Some works have pointed out the similarities and the differences between AIS and artificial neural networks: Dasgupta [69] and De Castro and Von Zuben [70]. De Castro has also used AIS to initialize the centers of radial basis function neural networks [71] and to produce a good initial set of weights for feed-forward neural networks[53].

Another interesting example of an hybrid AIS is the work of Hart and Ross [55], which combines sparse distributed memories and ideas from the NIS to cluster data in a dynamical changing environment.

2.3 Anomaly detection

In general, the problem of anomaly detection can be seen as a two class classification problem. Given an element from a given problem space, the system should classify it as normal or abnormal. However, this is a very general characterization since it can correspond to very different problems depending on the specific context where it is interpreted.

From a statistical point of view, the problem can be seen as that of outlier detection. According to Hawkins, an outlier is *"... an observation that deviates so much from other observations as to arouse suspicion that it was generated from a different mechanism"* [72]. A common statistical approach to solve this problem [73] is to build a statistical model of the normal and use it to determine if a given observation is an outlier or not; basically, if the probability of the observation being generated by the distribution of the normal observations is low, the observation is an outlier. A more complex approach can also model the outlier generation mechanism.

In the previous approach, the idea is to *clean* the data of any observation that can be classified as an outlier. Another possibility is to use methods that *accommodate* the outliers, i.e. methods that can produce good estimates or inferences even in the presence of outliers. This kind of methods belongs to a more general area of statistics called *robust statistics* [74, 75].

The outlier detection point of view implicitly assumes that the data is available at once for both normal data and outliers (which are possibly caused by errors in the data collection or by noise). The interpretation of anomaly detection that we are interested in, is situated in a most dynamic context. In this case, an anomaly is considered as a state of a given system that is not consistent with the normal behavior of this system. According to this, an anomaly detection system will perform a continuous monitoring of the system and an explicit classification of each state as normal or abnormal. Notice that the statistical modeling of the normal can be applied to this definition of anomaly detection, but the robust statistics approach cannot.

This type of interpretation of anomaly detection fits well with the problem of intrusion detection in computer security, fault detection in hardware, and novelty or surprise detection in time series. The most common approach to perform anomaly detection in computer security uses a statistical model [76, 77] to calculate the probability of occurrence of a given value; the lower the probability, the higher the possibility of an anomaly. The main problem about building a statistical model of normal is that it needs to make assumptions about the distribution properties of the monitored variables, which, in general, are not known. Another problem is that, in general, statistical approaches model individually different variables that represent the state of the system[4], and the anomalies may depend on interactions between the different variables.

Other approaches also build models to predict the future behavior of systems or processes based on the present and past states [78, 79, 44, 80, 81, 82]. Accordingly,

[4]Despite the possibility for using multi-variate distributions, the assumptions are too restrictive to be applied to real problems.

if the actual state of the system differs considerably from the predicted state, an anomaly alarm is raised. This kind of approach is investigated by Lane [83], who uses a hidden Markov model technique to model the interactions of an user with the computer. This model assigns probabilities to a sequence of actions. If this probability is very low, an alarm is raised.

Data mining techniques have also been applied to solve anomaly detection problems [84, 85, 86]. This approach has the advantages of dealing with large data sets and being able to garner useful knowledge (generally expressed in terms of rules). Lee et al. [86] applied a rule induction algorithm [87], frequent episode mining [88], and association rules mining [89] to the discovery of anomalies in audit data.

In other approaches, an anomaly is considered as a deviation from a set of normal states. This assumes that there is a notion of distance in this space that allows to measure deviations. Examples of this kind of approach are the works of Eskin et al. [90], which proposes a geometrical framework for unsupervised anomaly detection, and the work of Portnoy et al. [91], which uses a self -organizing map (a neural network architecture) to cluster the normal feature vectors.

The next section gives a formal description of the anomaly detection problem that we will address in the present work.

2.3.1 Anomaly detection problem definition

The purpose of anomaly detection is to identify states of a system as normal or abnormal. The states of a system can be represented by a set of features. Accordingly,

Definition 1. System state space. A state of the system is represented by a vector of features, $x^i = (x^i_1, ..., x^i_n) \in [0, 1]^n$. Each state is represented by a set $U \subseteq [0, 1]^n$. It includes the feature vectors corresponding to all possible states of the system.

These 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 $[0, 1]$.

Definition 2. Normal subspace (crisp characterization). A set of feature vectors, *Self*$\subseteq U$, represents the normal states of the system. Its complement is called *Non_Self* and is defined as *Non_Self* = *U* - *Self*. In some cases, we will define the *Self* (or *Non_Self*) set using its characteristic function $\chi_{self} : [0,1]^n \rightarrow \{0,1\}$

$$\chi_{self}(\overrightarrow{x}) = \begin{cases} 1 & \text{if } \overrightarrow{x} \in Self \\ 0 & \text{if } \overrightarrow{x} \in Non\_Self \end{cases}$$

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

Definition 3. Normal subspace (non-crisp characterization). The characteristic function of the normal (or abnormal) subspace is extended to take any value within the interval $[0,1] : \mu_{self} : [0,1]^n \rightarrow [0,1]$. 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.[5]

The non-crisp 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 non-crisp characterization to the crisp one by establishing a threshold:

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

Definition 4. Anomaly detection problem. Given a set of normal samples, $Self' \subseteq Self$, build a good estimate of the normal space characteristic function

[5]This definition is basically a fuzzy set specification. In fact, the function μ_{self} is a membership function.

χ_{self} (or μ_{self} in the non-crisp case). This function should be able to decide whether or not the observed state of the system is anomalous.

Chapter 3

The Effect of Binary Matching Rules in the Negative Selection Algorithm

A process that is of absolute importance for the NIS is the antibody-antigen matching[1], since it is the basis for recognition and selective elimination mechanism of foreign elements (see Subsection 2.1.1.3). Most of the AIS models implement this recognition process, but in different ways (see Section 2.2). Basically, antigens and antibodies are represented as strings of data that correspond to the sequence of aminoacids constituting proteins in the NIS. The matching of two strings is determined by a function that produces a binary output (match or not-match).

The binary representation is general enough to subsume other representations; after all, any data element, whatever its type is, is represented as a sequence of bits in the memory of a computer (though, how they are treated differ). In theory, any matching rule defined on a high-level representation can be expressed as a binary matching rule. However, in this work, we restrict the use of the term *binary matching*

[1]Antibodies are part of a class of proteins called antigen-binding receptors, which also includes T-cell receptors (see Subsection 2.1.1.3). Although the following discussion refers to antibodies, it also applies to T-cell receptors.

rule to designate those rules that take into account the matching of individual bits representing the antibody and the antigen.

Most works on the NS algorithm have been restricted to binary matching rules like r-contiguous [61, 44, 10]. The reason is that efficient algorithms that generate detectors (antibodies or T-cell receptors) have been developed, exploiting the simplicity of the binary representation and its matching rules [44]. On the other hand, AIS approaches inspired by the immune memory mechanism often use real vector representation for antibodies and antigens [18, 64], as this representation is more suitable for applications in learning and data analysis. The matching rules used with this real-valued representation are usually based on Euclidean distance, (i.e. the smaller the antibody-antigen distance, the more affinity they have).

The NS algorithm has been applied successfully to solve different problems; however, some unsatisfactory results have also been reported [92]. As it was suggested by Balthrop et al. [93], the source of the problem is not necessarily the NS algorithm itself, but the kind of matching rule used. The same work [93] proposed a new binary matching rule, r-chunk matching (Equation 3.2), which appears to perform better than r-contiguous matching.

The starting point of this chapter is to address the question: do the low-level representation and its matching rules affect the performance of NS in covering the non-self space? This chapter provides some answers to this issue. Specifically, it shows that binary matching possesses a low-level representation that is unable to capture the structure of even simple problem spaces. In order to justify our argument, we use some of the binary matching rules reported in the literature and study how they behave in a simple bi-dimensional real space. In particular, we study the shape of the areas covered by individual detectors and by a set of detectors generated by the NS algorithm.

3.1 Binary matching rules in the negative selection algorithm

The algorithmic problem of generating good detectors using NS varies with the type of representation space (continuous, discrete, hybrid, etc.), the detector representation, and the process that determines the matching ability of a detector.

A binary matching rule is a rule that is defined in terms of individual bit matchings of detectors and antigens represented as binary strings. In this section, some of the most widely used binary matching rules are presented.

3.1.1 *r*-contiguous matching

The first version of the NS algorithm [10] used binary strings of fixed length, and the matching between detectors and new patterns is determined by a rule called r-contiguous matching. The binary matching process is defined as follows: given $x = x_1x_2...x_n$ and a detector $d = d_1d_2...d_n$,

$$d \text{ matches } x \equiv \exists i \leq n - r + 1 \text{ such that } x_j = d_j \text{ for } j = i, ..., i + r - 1, \qquad (3.1)$$

that is, the two strings match if there is a sequence of size r where all the bits are identical. The algorithm works in a generate-and-test fashion, i.e. random detectors are generated; then, they are tested for self-matching. If a detector fails to match a self string, it is retained for novel pattern detection.

Subsequently, two new algorithms based on dynamic programming were proposed [44], the *linear* and the *greedy* NS algorithm. Similar to the previous algorithm, they are also specific to binary string representation and r-contiguous matching. Both algorithms run in linear time and space with respect to the size of the self set, though the time and space are exponential on the size of the matching sequence, r.

3.1.2 *r*-chunk matching

A new binary matching scheme called r-chunk matching was proposed by Balthrop et al. [61]. This matching rule subsumes r-contiguous matching, that is, any r-contiguous detector can be represented as a set of r-chunk detectors. The r-chunk matching rule is defined as follows: given a string $x = x_1x_2...x_n$ and a detector $d = (i, d_1d_2...d_m)$, with $m \leq n$ and $i \leq n - m + 1$,

$$d \text{ matches } x \equiv \ x_j = d_j \text{ for } j = i, ..., i + m - 1, \tag{3.2}$$

where i represents the position where the r-chunk starts. Some preliminary experiments [61] suggest that the r-chunk matching rule can improve the accuracy and performance of the NS algorithm.

3.1.3 Hamming distance matching rules

One of the first works that modeled NIS concepts in developing pattern recognition was proposed by Farmer et al. [6]. Their work proposed a computational model of the NIS based on the idiotypic network theory of Jerne [7], and compared it with the *learning classifier system* [94]. This is a binary model representing antibodies and antigens and defining a matching rule based on the Hamming distance. A Hamming distance based matching rule can be defined as follows: given a binary string $x = x_1x_2...x_n$ and a detector $d = d_1d_2...d_n$,

$$d \text{ matches } x \equiv \sum_i \overline{x_i \oplus d_i} \geq r, \tag{3.3}$$

where $\oplus$ is the exclusive-or operator, and $0 \leq r \leq n$ is a threshold value.

Different variations of the Hamming matching rule were studied, along with other rules like r-contiguous matching, statistical matching and landscape-affinity matching [62]. The different matching rules were compared by calculating the signal-to-noise ratio and the function-value distribution of each matching function when applied to

a randomly generated data set. The conclusion of the study was that the Rogers and Tanimoto (**R&T**) matching rule, a variation of the Hamming distance, produced the best performance. The R&T matching rule is defined as follows: given a binary string $x = x_1x_2...x_n$ and a detector $d = d_1d_2...d_n$,

$$d \text{ matches } x \equiv \frac{\sum_i \overline{x_i \oplus d_i}}{\sum_i \overline{x_i \oplus d_i} + 2\sum_i x_i \oplus d_i} \geq r, \tag{3.4}$$

where $\oplus$ is the exclusive-or operator, and $0 \leq r \leq 1$ is a threshold value.

It is important to mention that a good detector generation scheme for this kind of rules is not available yet, other than the exhaustive generate-and-test strategy [10].

3.2 Non-binary matching rules

In addition to the binary matching rules, other rules have also been proposed. The following are some example rules classified according to the representation space (U):

- $U = \Sigma^n$, where Σ is a finite alphabet.
  - r-contiguous and r-chunk rules can be easily extended to this case. A NS algorithm for r-contiguous that extends the *greedy* algorithm [44] was proposed in by Singh [95].
- $U = \{0, 1, ..., m\}^n$, where $m \in \mathbb{N}$ and $m > 1$
  - Representation of detectors as hypercubes in this n-dimensional discrete space. An element is matched by a detector, if it is contained in the respective hypercube [50].
  - Landscape-affinity matching was proposed by Harmer et al. [62]. The sequence of values of an element define a landscape. Two elements match if their respective landscapes are highly similar.

- $U = [0, 1]^n$

  - In Chapter 4, we propose a matching rule where the detectors are represented as hyper-rectangles in a n-dimensional continuous space. All the elements inside the hyper-rectangle are matched by a detector [21].

  - A detector, $d = (c, r)$, is defined by an element $c \in [0, 1]^n$ and a radius r. Any element x such that $distance(c, x) \leq r$ is considered to be matched by d. All the elements matched by a detector (c, r) constitute a hyper-sphere with center c and radius r. This representation is common in immune-network-based approaches [18, 9]. In Chapter 5, we will discuss its application in NS algorithms [25, 24].

3.3 Analyzing the shape of binary matching rules

Usually, the self/non-self space (U) used by the NS algorithm corresponds to an abstraction of a concrete problem space. For instance, if the problem at hand is to detect anomalies in a machine, the problem space corresponds to the space of features that describe the state of the machine at a given time. Each element in the problem space (e.g. a feature vector) is mapped to a corresponding element in U (e.g. a bit string).

A matching rule defines a relation between the set of detectors[2] and U. If this relationship is mapped back to the problem space, it can be interpreted as a relation of affinity between elements in this space. In general, it is expected that elements that are matched by the same detector have some common property (in the machine example, the probability of a detector matching two elements that represent similar states of the machine should be high). So, a way to analyze the ability of a matching rule to capture this 'affinity' relationship in the problem space is to take the subset of U corresponding to the elements matched by a specific detector, and map this subset

[2]In some matching rules, the set of detectors is same as U (e.g. r-contiguous matching). In other cases, it is a different set that usually contains or extends U (e.g. r-chunk matching).

back to the problem space. Accordingly, this set of elements in the problem space is expected to share some common properties.

In this section, we apply the approach described above to study the binary matching rules presented in Section 3.1. The problem space used corresponds to the set $[0, 1]^2$. One reason for choosing this problem space is that multiple problems in learning, pattern recognition, and anomaly detection can be easily expressed in an n-dimensional real-valued space. Also, it makes it easier to visualize the *shape* of different matching rules.

In this illustration, the self/non-self space is composed of binary strings of length 16. An element (x, y) in the problem space is mapped to the string $b_0, ..., b_7, b_8, ..., b_{15}$, where the first 8 bits encode the integer value $\lfloor 255 \cdot x + 0.5 \rfloor$ and the last 8 bits encode the integer value $\lfloor 255 \cdot y + 0.5 \rfloor$. Two encoding schemes are studied: conventional binary representation and Gray coding. Gray coding is expected to favor binary matching rules, since the codifications of two consecutive numbers only differs by one bit.

Figure 3.1 shows some typical shapes generated by different binary matching rules with different r values. Each figure represents the area (in the problem space) covered by one detector. In all cases, the detector was chosen to match the point (0.5,0.5) (1000000010000000 in binary notation).

The shapes generated by r-contiguous rule (Figure 3.1(a)) are composed by vertical and horizontal stripes that constitute a grid-like shape. The r-chunk rule generates similar, but simpler shapes (Figure 3.1(b)). In this case, the area covered is composed of vertical or horizontal sets of parallel strips. The orientation depends on the position of the r-chunk; if it is totally contained in the first eight bits, the strips are vertically going from top to bottom. If it is contained on the last eight bits, the strips are oriented horizontally. Finally, if it covers both parts, it has the shape shown in Figure 3.1(b).

The area covered by Hamming and R&T matching rules has a fractal-like shape, shown in Figure 3.1(c) and 3.1(d), i.e. it exhibits self-similarity. It is composed of

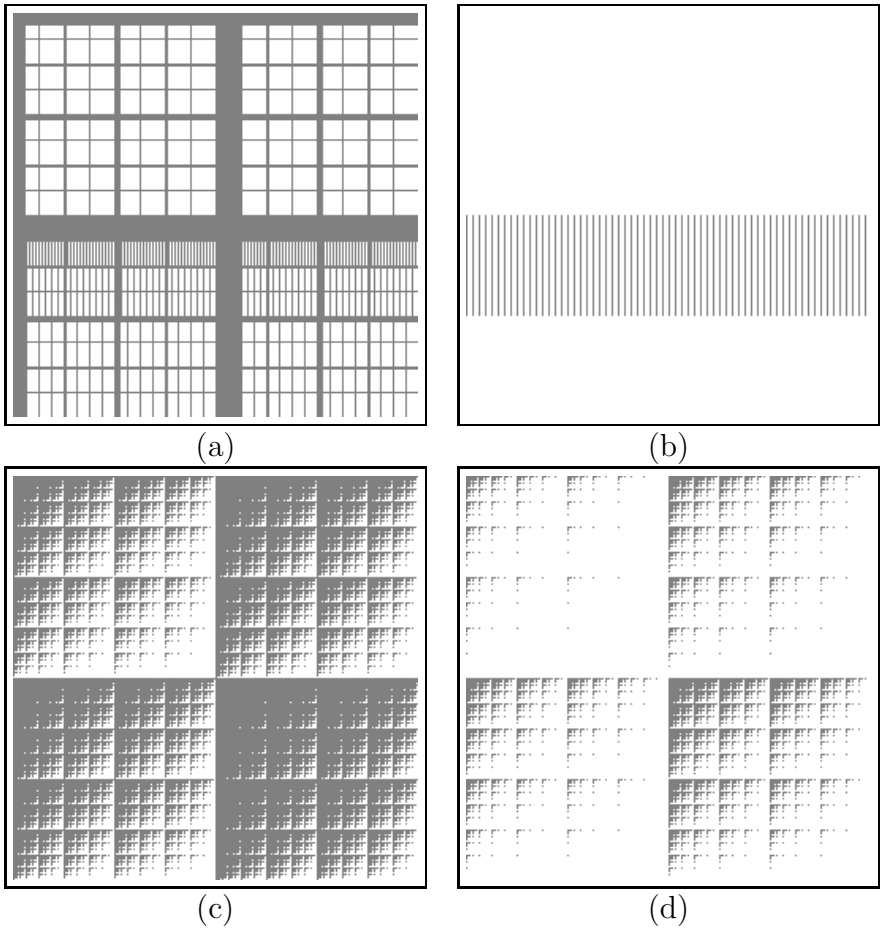


Figure 3.1: Areas covered in the problem space by an individual detector using different matching rules. The detector corresponds to 1000000010000000, which is the binary representation of the point (0.5,0.5). (a) r-contiguous matching, $r = 4$. (b) r-chunk matching, d = ****00001000****. (c) Hamming matching, $r = 8$. (d) R&T matching, $r = 0.5$.

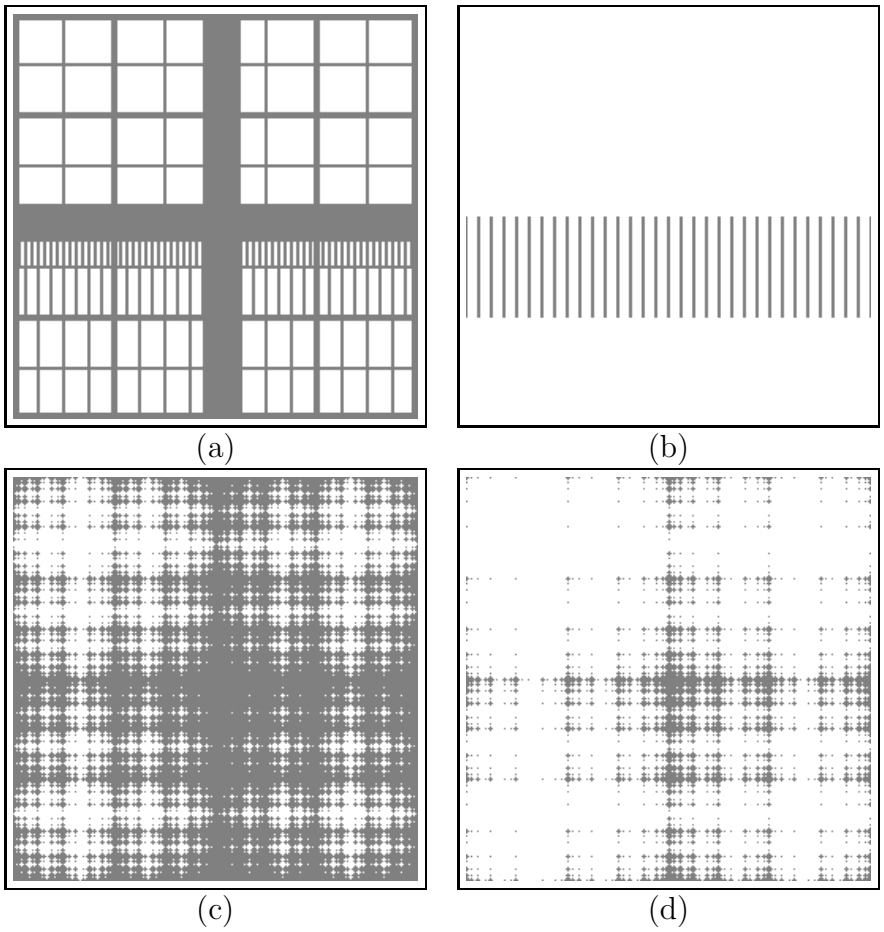


Figure 3.2: Areas covered in the problem space by an individual detector using Gray coding for the self/non-self space. The detector corresponds to 1100000011000000, which is the Gray representation of the point (0.5,0.5). (a) r-contiguous matching, $r = 4$. (b) r-chunk matching, $d = ******0011******$. (c) Hamming matching, $r = 8$. (d) R&T matching, $r = 0.5$.

points that have few interconnections. There is no significant difference between the shapes generated by the R&T rule and those generated by the Hamming rule, which is not a surprise, considering the fact that the R&T rule is based on Hamming distance.

The shape of the areas covered by r-contiguous and r-chunk matching is not affected by the change in codification from binary to Gray (as shown in Figures 3.2(a) and 3.2(b)). This is not the case with the Hamming and the R&T matching rule (Figures 3.2(c) and 3.2(d)). The reason is that the Gray encoding represents consecutive values with bit strings with small Hamming distance.

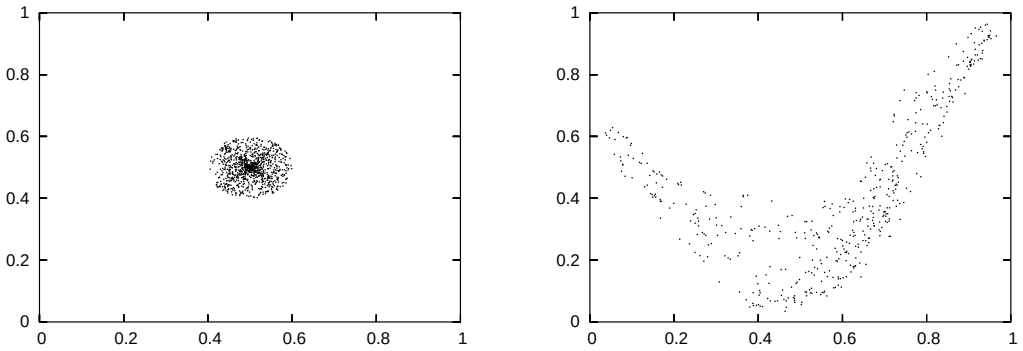


Figure 3.3: Self data sets used as input to the NS algorithm shown in a two-dimensional real problem space. (a) First data set composed of random points inside of a circle of radius 0.1. (b) Second data set corresponding to a section of the Mackey-Glass data set (described in Subsection 4.6.1.1).

In general, the shapes generated by the rules are sparse over the whole space. It is clear that the relation of proximity exhibited by these matching rules in the binary self/non-self space does not coincide with the natural relation of proximity in a real-valued, two-dimensional space. Intuitively, this seems to make the task harder of placing these detectors to cover the non-self space without covering the self set. This fact is further investigated in the next section.

3.4 Comparing the performance of binary matching rules

This section shows the performance of the binary matching rules (as presented in Section 3.1) in the NS algorithm. Experiments are performed using two synthetic data sets shown in Figure 3.3.

The first data set (Figure 3.3(a)) was created by generating random vectors in $[0, 1]^2$ with the center in (0.5,0.5) and scaling them to a norm less than 0.1, so that the points lies within a single circular cluster. The second set (Figure 3.3(b)) was extracted from the Mackey-Glass time series data set, which has been used in different works that apply AIS to anomaly detection problems [15, 25, 24] (for a more detailed

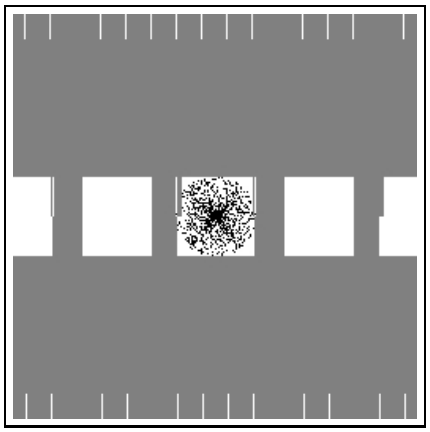


Figure 3.4: Coverage of space by a set of detectors generated by NS algorithm using r-contiguous matching (with $r = 7$). Black dots represent self-set points, and gray regions represent areas covered by detectors.

explanation of this data set, see Subsection 4.6.1.1). The original data set has four features extracted using a sliding window. We used only the first and the fourth features.

3.4.1 Experiments with the spherical cluster data set

Figure 3.4 shows a typical coverage of the non-self space corresponding to a set of detectors generated by the NS algorithm with r-contiguous matching for the first data set. The non-covered areas in the non-self space are known as *holes* [28] and are due to the characteristics of r-contiguous matching. In some cases, these holes can be good: since they are expected to be close to self strings, the set of detectors will not detect small deviations from the self set, making the NS algorithm robust to noise. However, when we map the holes from the representation (self/non-self) space to the problem space, they are not necessarily close to the self set, as shown in Figure 3.4. This result is not surprising; as we saw in the previous section (Section 3.3), the binary matching rules fail to capture the concept of *proximity* in this two-dimensional space.

We run the NS algorithm using different matching rules and varying the r value. Figure 3.5 shows the best coverage generated using standard (no Gray) binary rep-

resentation. The improvement in the coverage generated by r-contiguous matching (Figure 3.5(a)) is due to the higher value of r ($r = 9$), which produces more specific detectors. The coverage with the r-chunk matching rule (Figure 3.5(b)) is more consistent with the shape of the self set because of the high specificity of r-chunk detectors. The outputs produced by the NS algorithm with Hamming and R&T matching rules are the same. These two rules do not seem to do as well as the other matching rules (Figure 3.5(c)). However, by changing the encoding from binary to Gray (Figure 3.5(d)), the performance can be improved, since the Gray encoding changes the detector shape, as was shown in the previous section (Section 3.3). The change in the encoding scheme, however, does not affect the performance of the other rules for this particular data set.

The r-chunk matching rule produced the best performance in this data set, followed closely by the r-contiguous rule. This is due to the shape of the areas covered by r-chunk detectors which adapt very well to the simple structure of this self set, one localized, circular cluster.

3.4.2 Experiments with the Mackey-Glass data set

The second data set has a more complex structure than the first one, where the data is spread with a certain pattern. The NS algorithm should be able to generalize the self set with incomplete data. The NS algorithm was run with different binary matching rules, with both encodings (binary and Gray), and varying the value parameter r (the different values are shown in Table 3.1). Figure 3.6 shows some of the best results produced. Clearly, the tested matching rules were not able to produce a good coverage of the non-self space. The r-chunk matching rule generated satisfactory coverage of the non-self space (Figure 3.6(b)); however, the self space is covered by some lines resulting in erroneously detecting the self as non-self (false alarms). The Hamming-based matching rules generated an even more stringent result (Figure 3.6(d)) that covers almost the entire self space. The parameter r, which works as

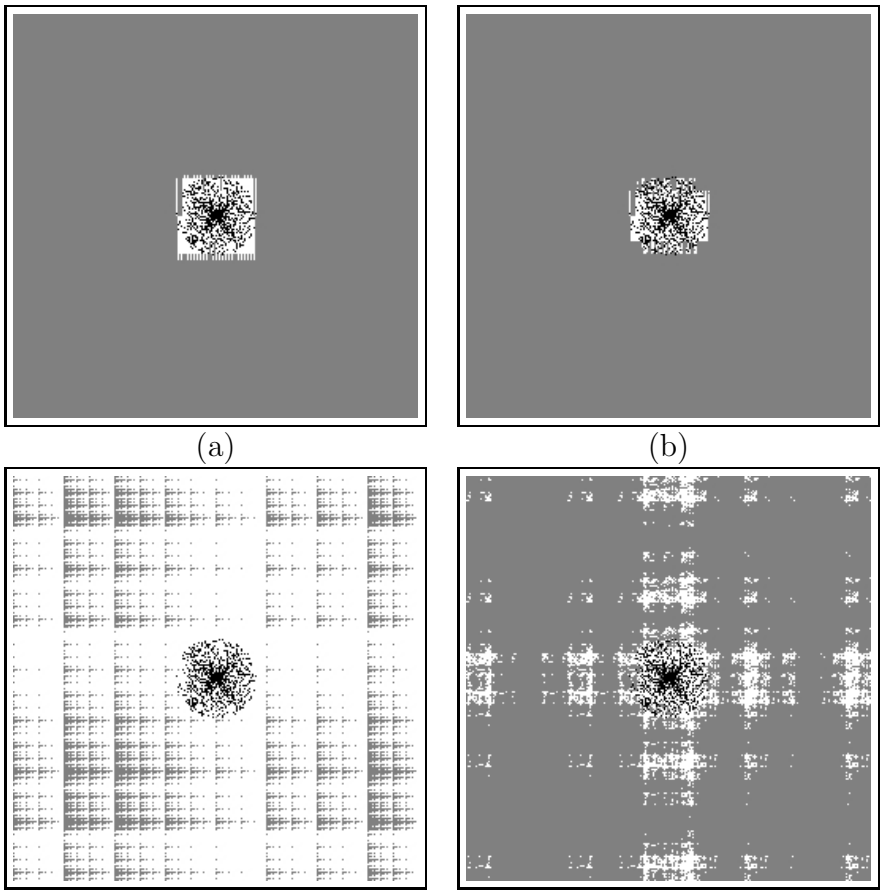


Figure 3.5: Best space coverage by detectors generated with NS algorithm using different matching rules. Black dots represent self-set points, and gray regions represent areas covered by detectors. (a) r-contiguous matching, $r = 9$, binary coding. (b) r-chunk matching, $r = 10$, binary coding. (c) Hamming matching, $r = 12$, binary coding (same as R&T matching, $r = 10/16$). (d) Hamming matching, $r = 10$, Gray coding (same as R&T matching, $r = 7/16$).

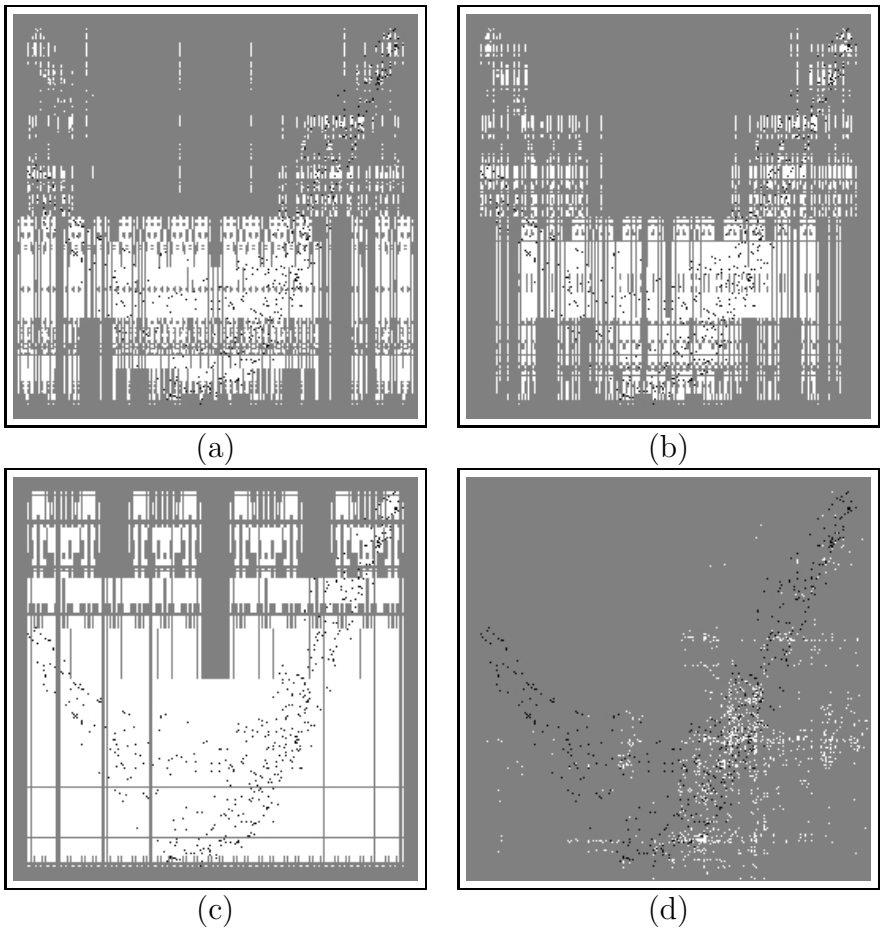


Figure 3.6: Best coverage of the non-self space by detectors generated with negative selection. Different matching rules, parameter values and codings (binary and Gray) were tested. (a) r-contiguous matching, $r = 9$, Gray coding. (b) r-chunk matching, $r = 8$, Gray coding. (c) r-chunk matching, $r = 7$, Gray coding. (d) Hamming matching, $r = 13$, binary coding (same as R&T matching, $r = 10/16$).

a threshold, controls the detection sensitivity. A smaller value of r generates more general detectors (i.e. covering a larger area) and decreases the detection sensitivity. However, for a more complex self set, changing the value of r from 8 (Figure 3.6(b)) to 7 (Figure 3.6(c)) generates a coverage with many holes in the non-self area, and still with some portions of the self covered by detectors. So, this problem is not with the setting of the correct value for r, but a fundamental limitation on of the binary representation that is not capable of capturing the semantics of the problem space. The performance of the Hamming-based matching rules is even worse; it produces a coverage that overlaps most of the self space (Figure 3.6(d)).

A better measure to determine the quality of the non-self space coverage with a set of detectors can be produced by matching the detectors against a test data set. The test data set is composed of both normal and abnormal elements as described in [25]. The results are measured in terms of the *detection rate* (percentage of abnormal elements correctly identified as abnormal) and the *false alarm rate* (percentage of the normal detectors wrongly identified as abnormal). An ideal set of detectors would have a detection rate close to 100%, while keeping a low false alarm rate. Table 3.1 accounts the results of all the performed experiments that combine different binary matching rules, different threshold or window size values (r), and two types of encoding. The results are also shown graphically in Figure 3.7 (binary encoding) and Figure 3.8 (Gray encoding). In general, the results are very poor. None of the configurations managed to deliver a good detection rate with a low false alarm rate. The best performance, which is far from good, is produced by the coverage depicted in Figure 3.6(b) (r-chunk matching, $r = 8$, Gray coding), with a detection rate of 73.26% and a false alarm rate of 47.47%. These results are in contrast with other previously reported [15, 95]; however, it is important to notice that the test set used on those experiments only contained abnormal data; so, no new normal data was presented during testing. In our case, the normal samples in the test data are, in general, different from those in the training set, though they are generated by the same process. Hence, the NS algorithm has to be able to generalize the structure of the self set in order to be able to classify correctly previously unseen normal patterns. But, is this a problem with the matching rule or a more general issue in the NS algorithm? In fact, the NS algorithm can perform very well on the same data set if the right matching rule is employed. We used a real value representation matching rule and followed the approach proposed in Chapter 5[24] on the second data set. The performance over the test data set was detection rate, 94%, false alarm, 3.5%. These results are clearly superior to all the results reported in Table 3.1.

Table 3.1: Results of different matching rules in NS using the the Mackey-Glass test data set. (r: threshold parameter, ND: number of detectors, D%: detection rate, FA%: false alarm rate).

		Binary			**Gray**		
	r	**ND**	**D%**	**FA%**	**ND**	**D%**	**FA%**
***r*-contiguous**	7	0			40	3.96%	1.26%
	8	343	15.84%	16.84%	361	16.83%	16.67%
	9	4531	53.46%	48.48%	4510	66.33%	48.23%
	10	16287	90.09%	77.52%	16430	90.09%	75.0%
	11	32598	95.04%	89.64%	32609	98.01%	90.4%
***r*-chunk**	4	0			2	0.0%	0.75%
	5	4	0.0%	0.75%	8	0.0%	0.75%
	6	18	3.96%	4.04%	22	3.96%	2.52%
	7	98	14.85%	16.16%	118	18.81%	13.13%
	8	549	54.45%	48.98%	594	73.26%	47.47%
	9	1942	85.14%	72.97%	1959	88.11%	67.42%
	10	4807	98.01%	86.86%	4807	98.01%	86.86%
	11	9948	100%	92.92%	9948	100%	92.92%
	12	18348	100%	94.44%	18348	100%	94.44%
Hamming	11	0			0		
	12	1	0.99%	3.03%	7	10.89%	8.08%
	13	2173	99%	91.16%	3650	99.0%	91.66%
	14	29068	100%	95.2%	31166	100%	95.2%
Rogers & Tanimoto	8/16	0			0		
	9/16	1	0.99%	3.03%	7	10.89%	8.08%
	10/16	2173	99%	91.16%	3650	99%	91.66%
	11/16	29068	100%	95.2%	31166	100%	95.2%
	12/16	29068	100%	95.2%	31166	100%	95.2%

3.5 Summary

In this chapter, we discussed different binary matching rules used in the negative selection (NS) algorithm. The primary applications of NS have been in the field of change (or anomaly) detection, where the detectors are generated in the complement space which can detect changes in data patterns. The main component of NS is the choice of a matching rule, which determines the similarity between two patterns in order to classify self/non-self (normal/abnormal) samples. There exists a number of matching rules and encoding schemes for the NS algorithm. This chapter examines the properties (in terms of coverage and detection rate) of each binary matching rule for different encoding schemes.

Experimental results showed that the studied binary matching rules cannot produce a good generalization of the self space, which results in a poor coverage of the non-self space. The reason is that the affinity relation implemented by the matching rule at the representation level (self/non-self) space cannot capture the affinity relationship at the problem space. This phenomenon is observed in our experiments with a simple real two-dimensional problem space.

The main conclusion of this chapter is that the matching rule for NS algorithm needs to be chosen in such a way that it represents accurately the affinity relationship in the problem space. Another factor to take into account is the type of application. For instance, in change detection applications, where the complete knowledge of the self space is available, the poor generalization capabilities of binary matching rules does not seem to be a major issue. In contrast, in anomaly detection applications, like those in computer security where we cannot expect ever to have a complete training set, it is crucial to count on matching rules that can capture the semantics of the problem space.

The results shown in this chapter provide a justification to explore new representation and matching rules for the NS algorithm; this is the main goal of the subsequent chapters. Particularly, our effort is directed to investigate methods to generate good

sets of detectors in real valued spaces. This type of representation also opens the possibility to integrate NS with other AIS techniques like those inspired by the immune memory mechanism [18, 64].

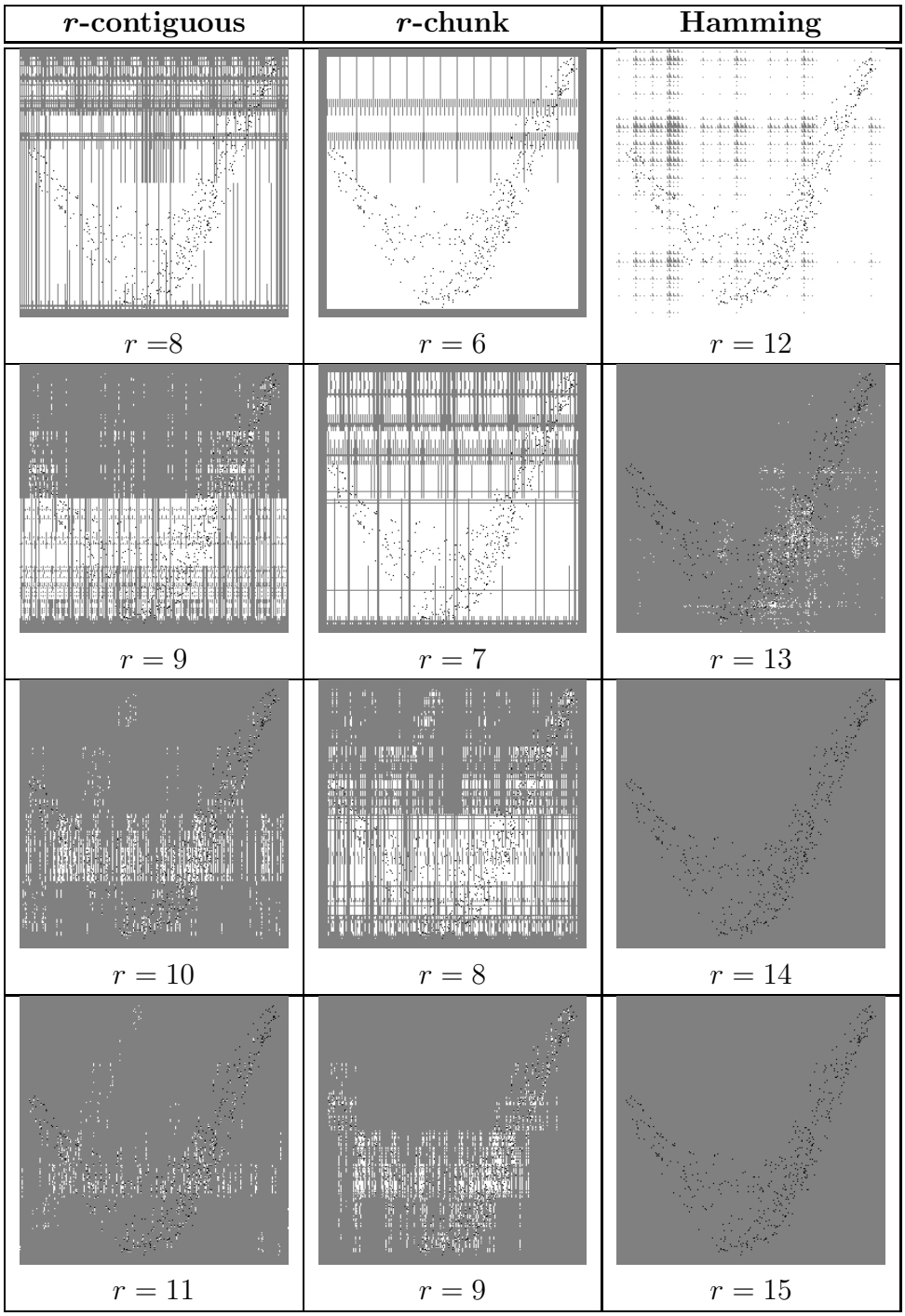


Figure 3.7: Coverage of the non-self space by detectors generated with negative selection. Different matching rules and parameter values were tested using binary encoding. The results using Hamming maytching rule are same as the results using R&T rule.

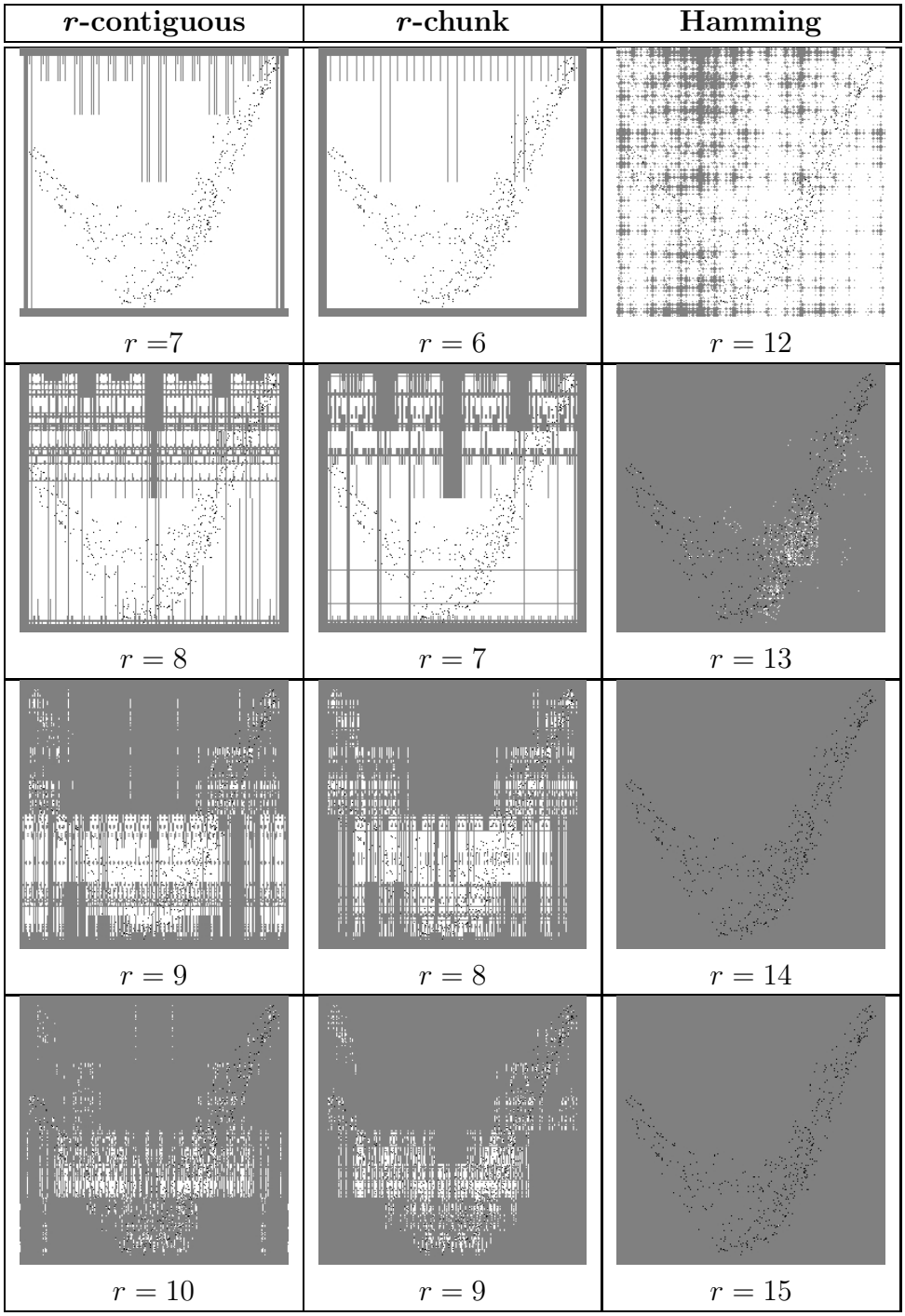


Figure 3.8: Coverage of the non-self space by detectors generated with negative selection. Different matching rules and parameter values were tested using Gray encoding. The results using Hamming maytching rule are same as the results using R&T rule.

Chapter 4

Negative Selection with Detection Rules

4.1 Introduction

In the previous chapter, it was shown that binary matching rules have limitations in terms of capturing the semantics of some complex self/non-self spaces. Additionally, there are other issues found to exist that have prevented the NS algorithm from being applied more extensively:

- Scalability: in order to guarantee good levels of detection, a large number of detectors has to be generated (depending on the size of the self). For some problems, the number of detectors could be unmanageable [92].

- The low-level detector representation prevents the extraction of meaningful domain knowledge. This makes it difficult to analyze reasons for reporting an anomaly.

- A sharp distinction exists between the normal and abnormal. This divides the space into two subsets: *self* (the normal) and the *non-self* (abnormal). An element in the space is considered to be abnormal if there exists a detector that matches it. In reality, the 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, like binary, makes it difficult to integrate the NS algorithm with other immune algorithms.

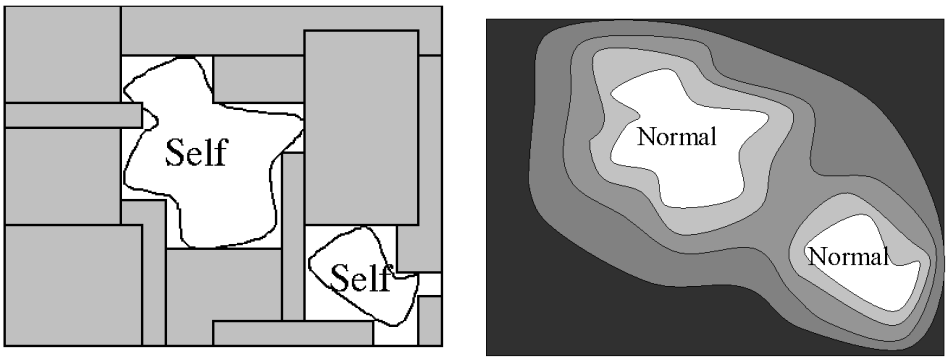
In this chapter, we propose a higher level representation for the detectors (antibodies) that allows the extraction of knowledge through the application of NS algorithms. Specifically, the self/non-self space corresponds to a subset of $\mathbb{R}^n$, the unitary hypercube $[0, 1]^n$, and the detectors are hyper-rectangles contained in this space, which can be interpreted as anomaly detection rules. The added structure to detectors necessitates the use of a more sophisticated detector generation algorithm; we used an evolutionary algorithm for this purpose.

An additional issue this work tries to address is the crisp distinction between self and non-self. The proposed algorithm divides the non-self space into different levels. The idea is that the technique not only detects anomalous samples, but also estimates the amount of deviation from the normal.

The proposed algorithm was first presented in 2002, where it was used to detect anomalies in network traffic [21]. The algorithmic details are presented in Section 4.2. Experimental results are reported in Section 4.3. Additionally, an improved detector evolution algorithm [22] is introduced in Section 4.4. We extended the detector representation to support fuzzy anomaly detection rules [23]. This work is presented in Section 4.5.

4.2 Negative selection with detection rules (NSDR)

Instead of using binary encoding for the negative selection algorithm [10], our approach uses real-valued representation to characterize the self/non-self space and



(a) (b)

Figure 4.1: Self/non-self space. (a) Approximation of the non-self space by rectangular interval rules. (b) Levels of deviation from the normal in the non-self space.

evolves a set of detectors that can cover the (non-self) complementary subspace (as shown in Figure 4.1). The basic structure of these detection rules is as follows:

R^1: If $Cond_1$ then **non_self**

$\vdots$ $\vdots$ $\vdots$

R^m: If $Cond_m$ then **non_self**

where,

- $Cond_i = x_1 \in [low_1^i, high_1^i]$ and $\ldots$ and $x_n \in [low_n^i, high_n^i]$
- $(x_1, ..., x_n)$ is a feature vector
- $[low_i^j, high_i^j]$ specifies the lower and upper values for the feature x_i in the condition part of the rule R^j.

The condition part of each rule defines a hypercube in the self/non-self space, $[0.0, 1.0]^n$. Then, a set of these rules tries to cover the non-self space with hypercubes. For the case $n = 2$, the condition part of a rule represents a rectangle. Figure 4.1(a) illustrates an example of this kind of cover for $n = 2$.

The non-self characteristic function (crisp version, see Subsection 2.3.1) generated by a set of rules $R = \{R^1, ..., R^m\}$ is defined as follows:

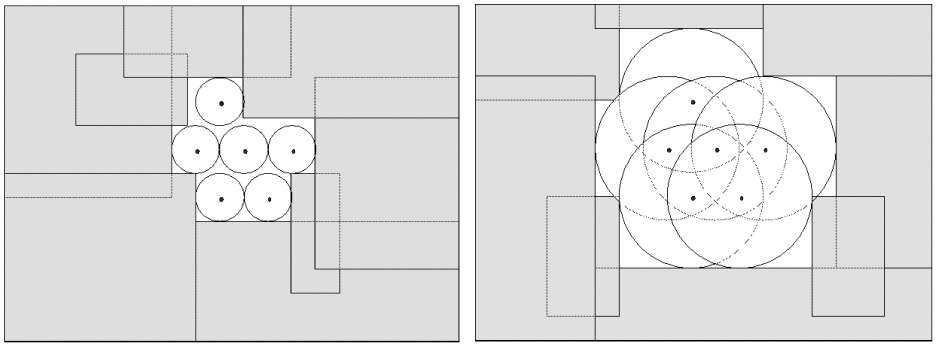
$$\chi_{non\_self,R}(\overrightarrow{x}) = \begin{cases} 1 & \text{if } \exists R^j \in R \text{ such that } \overrightarrow{x} \in R^j \\ 0 & \text{otherwise} \end{cases}$$

We used a genetic algorithm to evolve rules to cover the non-self 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 genetic algorithm.

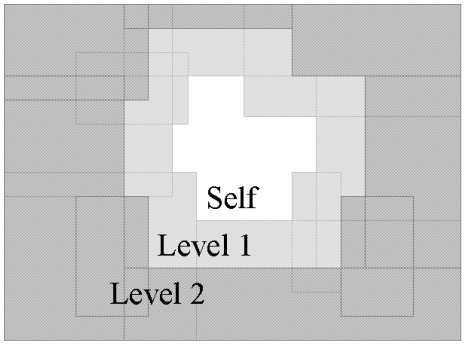
As was described previously, a good characterization of the abnormal (non-self) space should be non-crisp. Then, the non-self space can further be divided in different levels of deviation. In Figure 4.1(b), these levels of deviation are shown as concentric regions around the self zones.

In order to characterize 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 value of v represents less variability (a smaller self space). Figure 4.2 shows two sets of rules that characterize self spaces with a large and small value of v. Figure 4.2(a) shows a covering using a small variability parameter v. Figure 4.2(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. Figure 4.2(c) shows the levels of deviation defined by two coverings.

In the non-self space, we use a genetic algorithm with different values of v to generate a set of rules that can provide complete coverage. A set of rules looks like:



(a) (b)



(c)

Figure 4.2: 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 non-self (abnormal) space using a small value of v. (b) Rectangular rules cover the non-self space using a large value of v. (c) Level of deviation defined by each v, where level 1 corresponds to non-self cover in (a) and level 2 corresponds to non-self cover in (b).

$$
\begin{array}{rlll}
R^1\text{: If} & Cond_1 & \text{then} & \textbf{Level 1} \\
\vdots & \vdots & & \vdots \\
R^i\text{: If} & Cond_i & \text{then} & \textbf{Level 1} \\
R^{i+1}\text{: If} & Cond_{i+1} & \text{then} & \textbf{Level 2} \\
\vdots & \vdots & & \vdots \\
R^j\text{: If} & Cond_j & \text{then} & \textbf{Level 2} \\
\vdots & \vdots & & \vdots
\end{array}
$$

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

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

where $level(R^j)$ represents the deviation level reported by the rule R^j.

4.2.1 Genetic algorithm in detection rule generation

The genetic algorithm attempts to evolve 'good' rules [96, 97, 98] that cover the non-self 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 multi-objective, multi-modal optimization problem. We are not interested in one solution but a set of solutions that collectively can solve the problem (covering of the non-self region).

A niching technique is used with GAs to generate different rules. The input to the GA is a set of feature vectors $S' = \{x^1, ..., x^l\}$, which are normal behavior samples. Each element x^j in S' is an n-dimensional vector $x^j = (x^j_1, ..., x^j_n)$.

The algorithm for the rule generation is shown in Figure 4.3, where

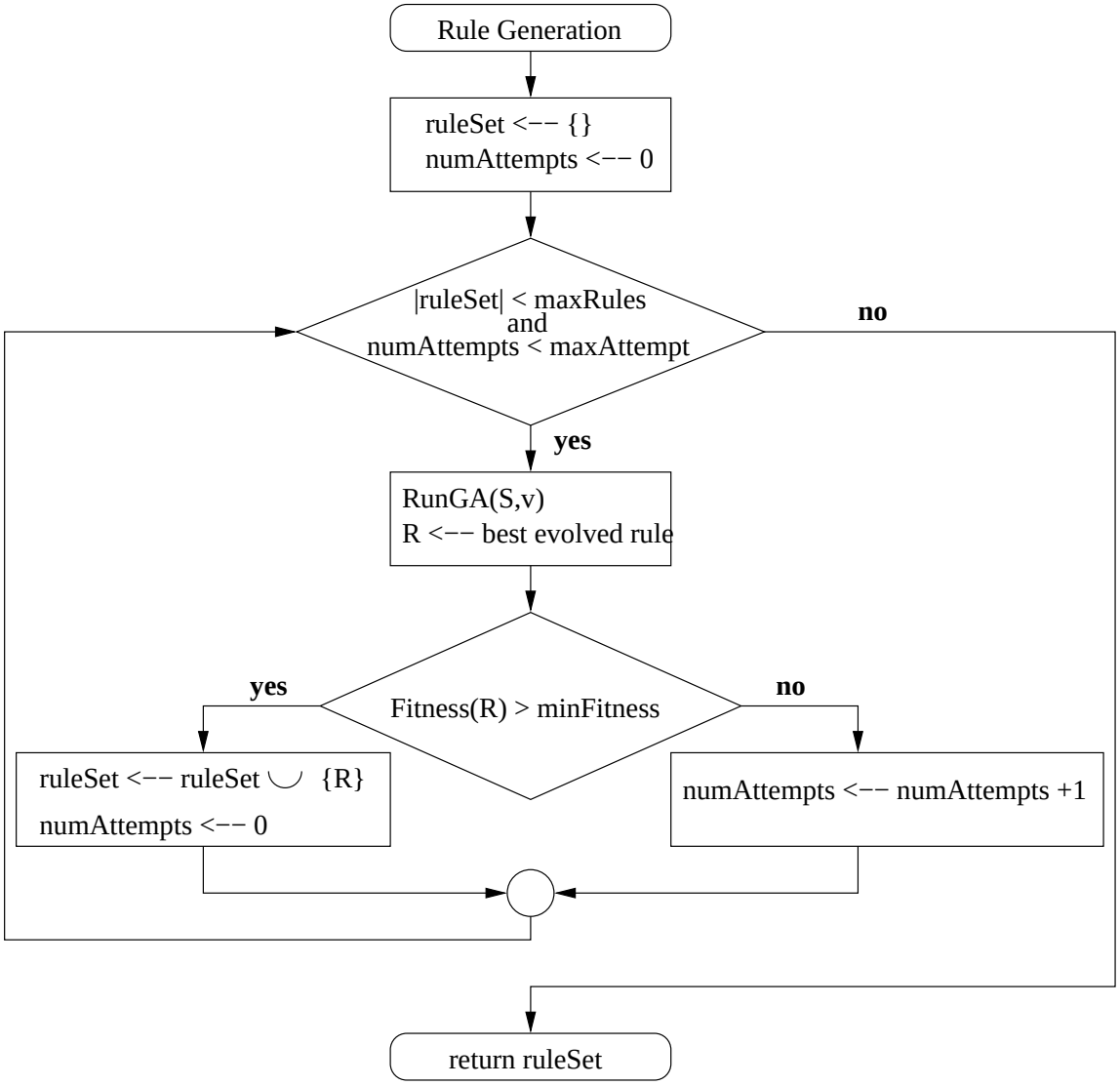


Figure 4.3: NSDR rule generation using a genetic algorithm with sequential niching.

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.*

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 $O(num\_gen \cdot pop\_size \cdot f_{time})$, 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 (see Section 4.2.1.2). Therefore, the execution time of the NSDR algorithm is $O(m \cdot num\_gen \cdot pop\_size \cdot |S'|)$, where m is the number of generated rules.

4.2.1.1 Chromosome representation

Each individual (chromosome) in the genetic algorithm represents the condition part of a rule, since the consequent part is the same for all the rules (the descriptor belongs to the non-self). However, the levels of deviation in the non-self 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.

4.2.1.2 Fitness evaluation

Given a rule R with a condition part ($x_1 \in [low_1, high_1]$ **and** $\ldots$ **and** $x_n \in [low_n, high_n]$), we say that a feature vector $x^j = (x_1^j, ..., x_n^j)$ 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:

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

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

- The volume of the subspace represented by the rule:

$$volume(R) = \prod_{i=1}^{n}(high_i - low_i)$$

The raw fitness is defined as:

$$raw\_fitness_R = volume(R) - C \cdot num\_elements(R)$$

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 bigger the coefficient (C), the higher the imposed penalty. The raw fitness can also take negative values.

Since the coverage of the non-self 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.

4.2.1.3 Sequential niching algorithm

The idea is to run the GA multiple times [99] to generate different rules so as to cover the entire non-self region. In each run, we want to generate a new rule, that is, a rule that can cover a portion of the non-self region. The raw fitness of each rule is modified according to the overlap with the previously chosen rules. The following pseudo-code segment shows how the final fitness of the rule R is calculated.

$fitness_R \leftarrow raw\_fitness_R$

for each $R^j \in ruleSet$ do

  $fitness_R \leftarrow raw\_fitness_R - volume(R \cap R^j)$

end-For

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

4.3 NSDR using sequential niching experimentation

In this section, we test the proposed approach with network traffic data. The idea is to examine if the system is able to detect some attacks subsequently, when the system is trained with normal traffic patterns.

In order 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. This technique is described in Section 4.3.2.

4.3.1 Data set (MIT-Darpa 99)

This data set is a version of the 1999 DARPA intrusion detection evaluation data set generated and managed by MIT Lincoln Labs [100]. 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 contain complete weeks with normal data (not mixed with attacks). This provides enough samples to train the detection system.

The 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, and the others were inside attacks. Only the network attacks were considered for our testing. These attacks are described in Table 4.1 and the attack time-line is shown in Figure 4.4.

Table 4.1: 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

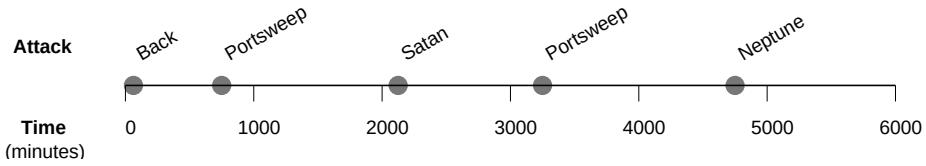


Figure 4.4: Network attacks on the second week.

Three parameters were selected to detect some specific type of attacks. These parameters were sampled each minute (using *tcpstat*) and normalized. Table 4.2 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 $R = \{r_1, r_2, ..., r_n\}$ in an overlapping sliding window fashion:

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

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

Table 4.2: 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

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 9-dimensional feature vectors was generated.

4.3.2 Positive characterization (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[1] 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 *Non_Self* set, since its definition is more natural, and the derivation of the *Self* set characteristic function is straightforward.

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

Here, $d(x, s)$ is a Euclidean distance metric (or any Minkowski metric[2]). $D(\overrightarrow{x}, Self)$ is the nearest-neighbor distance, that is, the distance from x to the closest point in *Self*. Then, the closer an element $\overrightarrow{x}$ is to the self set, the closer the value of $\mu_{non\_self}(\overrightarrow{x})$ is to 0.

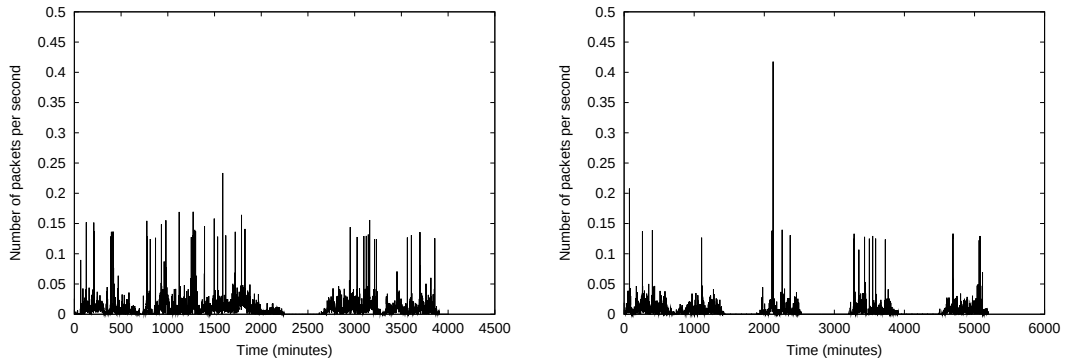
The crisp version of the characteristic function is the following:

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

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)

[1] This approach is known as lazy learning or instance based learning [101]. It is used commonly in classification algorithms.

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



(a) training set (S2)

(b) testing set (T2)

Figure 4.5: 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.

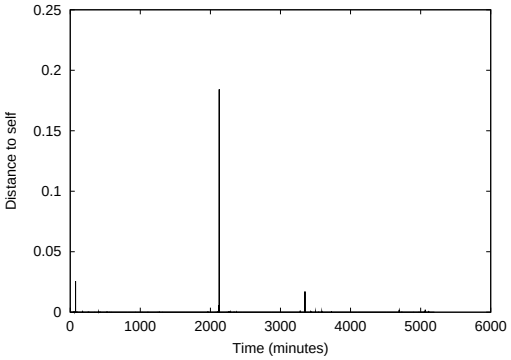
represents the amount of allowable variability in the self space (the maximum distance that a point can be from the *Self* samples to be considered as normal).

This positive characterization can be implemented efficiently using spatial trees. In our implementation, a KD-Tree [102, 103, 104] 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: 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) where the points are stored.

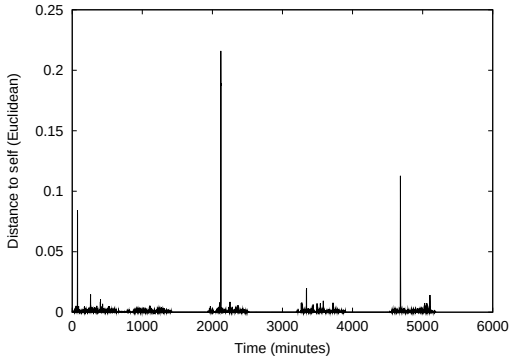
This representation allows answering queries in an efficient way. The amortized cost of a nearest-neighbor query is *O(log N)* [103]. We used a library (that implements the KD-Tree structure) developed at the University of Maryland [105].

4.3.2.1 Positive characterization experiments

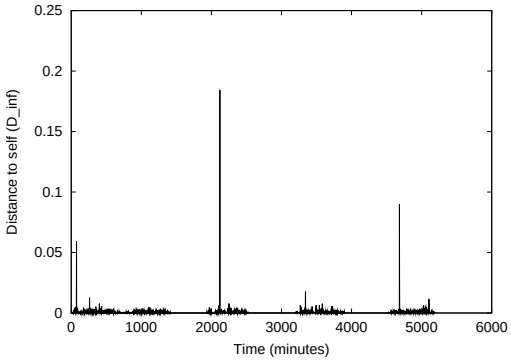
In each experiment, the training set was used to build a KD-tree to represent the self set. Then, the distance (nearest neighbor) from each point in the testing set to the self set was measured to determine deviations.



(a) w =1



(b) w=3, Euclidean



(c) w=3, Dinf

Figure 4.6: Distance from the testing set (T2) to the self set (S2) ($\mu_{non\_self}(\overrightarrow{x})$). (a) Using window size 1. (b) Using window size 3 and Euclidean distance. (c) Using window size 3 and D_∞ distance.

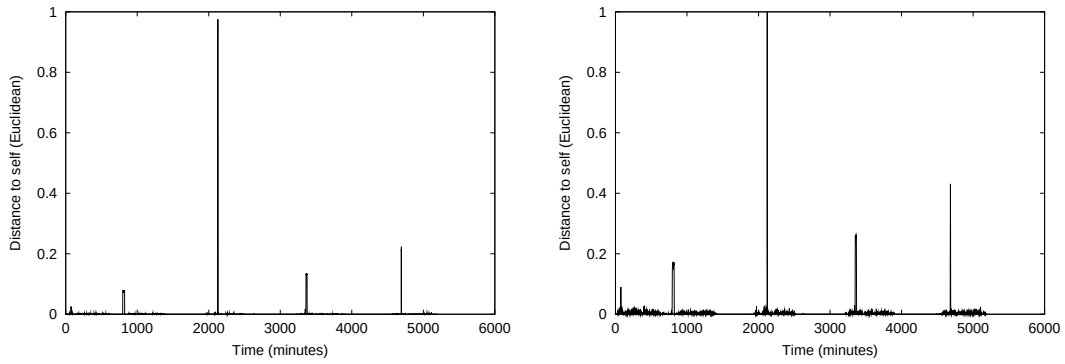
For this set of experiments, the variables were considered independently; that is, the feature vectors were built using only one variable (time series) each time. Figure 4.5 shows an example of the training and testing data sets for the parameter *number of packets per second*. Figure 4.6(a) represents the non-self characteristic function $\mu_{non\_self}(\overrightarrow{x})$, 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. Figures 4.6(b) and 4.6(c) show $\mu_{non\_self}(\overrightarrow{x})$ for using a window size of 3. In Figure 4.6(b), the Euclidean distance is used, and in Figure 4.6(c), the D_∞ distance is used.

The plots (in Figure 4.6) of the non-self 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 4.1 and Figure 4.4). We conclude the following from these results:

- Using only one parameter is not enough to detect all five attacks. Figure 4.6 shows how the function $\mu_{non\_self}(\overrightarrow{x})$ detects deviations that correspond to attacks; however, none of the parameters is able to detect, independently, 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 (Figure 4.6(a)) to 3 (Figures 4.6(b) and 4.6(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.



(a) w = 1 (b) w = 3

Figure 4.7: Distance from test sets to the self set ($\mu_{non\_self}(\overrightarrow{x})$) using $S1$,$S2$,$S3$. (a) Window size 1. (b) Window size 3.

- The change of the distance metric from Euclidean (Figure 4.6(b)) to D_∞ (Figure 4.6(c)) does not modify the number and type of the 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 in order to test whether the PC technique can detect all the attacks. Accordingly, we performed two experiments by varying the sliding window size:

- 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.

- 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.

Figure 4.7 shows the non-self characteristic function for feature vectors conformed with samples of three time series. In all cases, there are five remarkable anomalies that correspond to five attacks. Like previous experiments, an increase in 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 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.

According to Definition 3 (Subsection 2.3.1), the crisp version of the anomaly detection function $\mu_{non\_self}(\overrightarrow{x})$ 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 on the number of detections, but also increasing the number of false positives (false alarms). In order to show this trade-off between the false alarm rate and the detection rate, **ROC** (receiver operating characteristics) diagrams [106] are drawn. The anomaly detection function $\mu_{non\_self,t}(\overrightarrow{x})$ 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. The detection and false alarm rates are calculated using the following equations:

$$\text{Detection rate} = \frac{TP}{TP + FN} \tag{4.1}$$

$$\text{False alarm rate} = \frac{FP}{TN + FP}, \tag{4.2}$$

where:

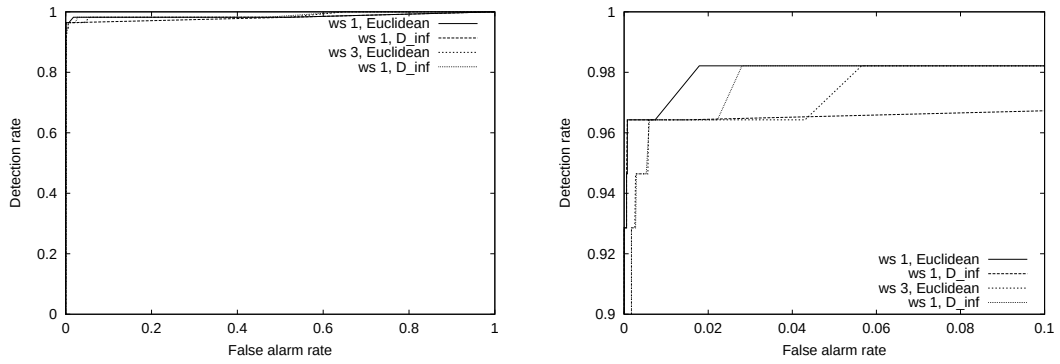
TP : true positives – anomalous elements identified as anomalous;

TN : true negatives – normal elements identified as normal;

FP : false positives – normal elements identified as anomalous; and

FN : false negatives – anomalous elements identified as normal.

The Figure 4.8 shows the ROC diagrams for the $\mu_{non\_self}(\overrightarrow{x})$ functions shown in Figure 4.7. 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



(a) (b)

Figure 4.8: ROC diagrams for the $\mu_{non\_self}(\overrightarrow{x})$ function shown on Figure 4.7. (a) Full scale. (b) Detail of the upper-left corner.

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 the larger window size was able to detect it.

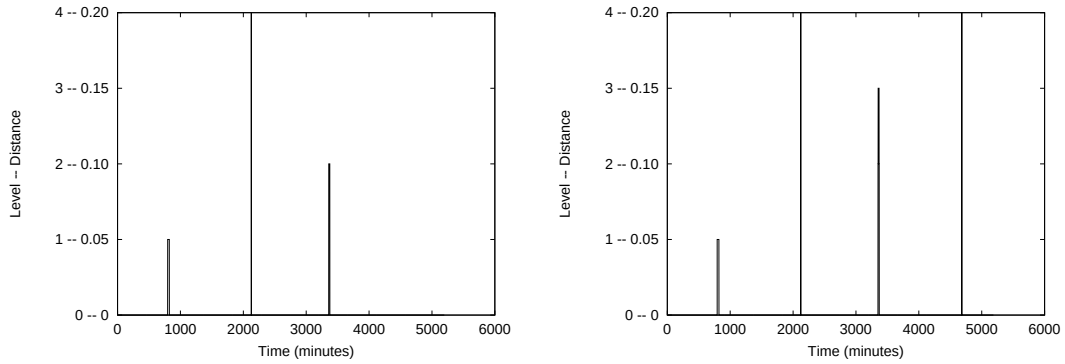
The positive characterization technique has shown to work well on the performed experiments. The main drawback of this technique is its memory requirements, since it is necessary to store the samples that constitute the normal profile. The amount of data generated by network traffic can be large, making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

4.3.3 Experimentation and results

In order to test the negative characterization approach (NSDR), we used the MIT-Darpa 99 data set [100] (mentioned in Section 4.3.2.1). 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).

Table 4.3: 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) w =1 (b) w = 3

Figure 4.9: Indicates the deviations in the testing set detected by the evolved rule set. (a) For window size 1. (b) For window size 3.

The parameters for the genetic algorithm 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 genetic algorithm 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 10 times and the results reported correspond to the average of these runs.

Table 4.3 shows the number of rules generated by the genetic algorithm 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.

Table 4.4: Best true positive rates for the different techniques with a maximum false alarm rate of 1%

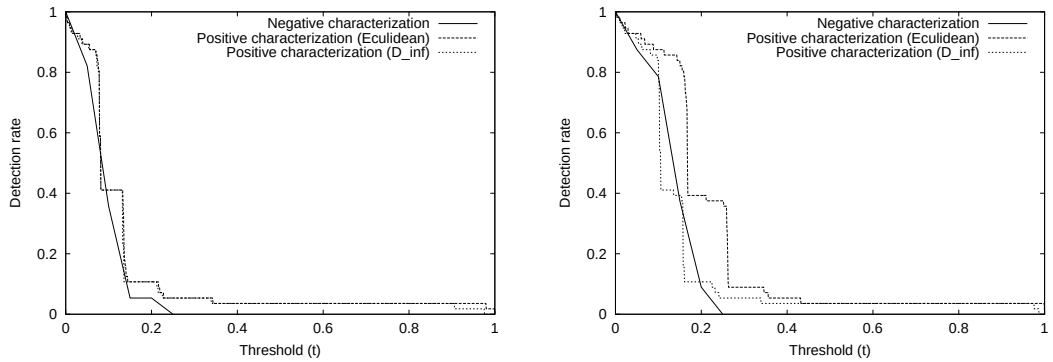
Detection Technique	**Window size 1**	**Window size 3**
Positive Characterization (Euclidean)	92.8%	96.4%
Positive Characterization (D_∞)	92.8%	92.8%
Negative Characterization	82.1%	87.5%

Figure 4.9 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, while with a window size of 3, four out of five attacks are detected.

The negative characterization technique (NSDR) is more efficient (in time and space) compared to the positive characterization (PC). In the case of a window size of 1, the PC needs to store 5202*3=15,606 floating-point values; the NSDR technique only has to store 4*6 = 24 floating points 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 4.10 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 the NSDR one 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 4.4 summarizes the best true positive rates (with a maximum false alarm of 1%) accomplished by the two techniques.

Esponda et al. [107] suggested that this comparison between the PC technique and the NSDR method is not meaningful since 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.

[3]The number of floating point numbers needed by the positive characterization is equal to (5192 samples)*(9 dimensions) = 46,728. The number of floating points numbers needed by the negative characterization is (94 rules)*(18 floating values per rule) = 1,698.



(a) w = 1 (b) w = 3

Figure 4.10: Comparison of the true positives rate of the detection function $\mu_{non\_self,t}(\overrightarrow{x})$ generated by positive characterization (PC) and negative characterization (NSDR) for different values of t. (a) Window size 1. (b) Window size 3.

As was mentioned before, the proposed NSDR technique produces a good estimate of the levels of deviation. In order to evaluate this estimate, a detailed comparison of the NSDR output levels and PC distance range was performed. The results are illustrated in Table 4.5 in the form of a confusion matrix. For each element in the testing set, the function $\mu_{non\_self}(\overrightarrow{x})$ 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 4.5 suggest that the NSDR approach better approximates the deviation reported by PC using D_∞ distance. To support this claim precisely, we measured the number of testing samples for the all 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 4.6.

Table 4.5: Confusion matrix for PC and NSDR reported deviations. The values of the matrix elements correspond to the number of testing samples in each class. The diagonal values represent correct classification

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

Table 4.6: The difference between PC and NSDR reported levels for test data set. 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 NSDR are the same, a difference of one means that the results differ by 1 level, etc.

Difference between PC and NSDR 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%

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% of the outputs from the NSDR algorithm are same as the PC algorithm, 100% of the NSDR outputs are in the range of 0 or 1 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 (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.

4.3.4 Analysis of results

We investigated genetic algorithms to evolve detectors in the complement pattern space to identify any changes in the normal behavior of monitored behavior patterns. This technique (NSDR) is used to characterize and to identify different intrusive activities by monitoring network traffic, compared with another approach (PC). We used a real-world data set (MIT-Lincoln Lab) that has been used by other researchers for testing different approaches. The following are some preliminary observations:

- 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 positive characterization (with a detection rate of 87.5% and a maximum false alarm rate of 1%).

- The best results were produced when we used window size of 3. We observed that a bigger window size makes the system more sensitive to deviations.

4.4 NSDR using deterministic crowding

The purpose of the genetic algorithm is to evolve 'good' rules to cover the non-self space. In general, one rule is not enough; instead, a set of rules that solve the problem cooperatively is necessary. In Section 4.2, we used a genetic algorithm combined with a sequential niching technique [99]. That approach was useful in evolving good detection rules. The main drawback of that approach is that the genetic algorithm must be run multiple times to generate multiple rules. The approach proposed in this section uses a niching technique, deterministic crowding [108], that allows the generation of multiple rules in a single run.

4.4.1 The algorithm

The NSDR algorithm using deterministic crowding (DC) is shown in Figure 4.11. The main inputs to the algorithm are a set of n-dimensional feature vectors $S = \{x^1, ..., x^l\}$, 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 $\{v_1, ..., v_{numLevels}\}$. Additional parameters to the algorithm are the population size (pop_size) and the number of generations (num_gen).

The execution time of this algorithm is $O(num\_levels \cdot num\_gen \cdot pop\_size \cdot |S'|)$, where $|S'|$ is the number of self samples and is included in the expression since the time complexity of the fitness calculation is $O(|S'|)$. 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 sequential niching (see Subsection 4.2.1).

The chromosome representation and the fitness function used are the ones described in Subsections 4.2.1.1 and 4.2.1.2, respectively.

```
NS-DETECTOR-RULES(S', num_levels, {v_1, ..., v_numLevels})
                S'  :  set of self samples
        num_levels  :  number of deviation levels
{v_1, ..., v_numLevels}  :  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 4.11: Negative selection with detection rules (NSDR) algorithm using deterministic crowding (DC).

4.4.1.1 Individual's distance calculation

A good measure of distance between individuals is important for deterministic crowding niching, since 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.

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.

4.4.2 NSDR using deterministic crowding experimentation

The proposed algorithm was tested with the data set presented in Section 4.3.1. We used as the training set the time series $S1$, $S2$ and $S3$, and as the testing set the 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 genetic algorithm were population size 200, number of generations 2000, mutation rate 0.1, and coefficient of sensitivity 1.0 (high sensitivity).

The genetic algorithm 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 10 times and the results reported correspond to the average of these runs.

Table 4.7: Number of generated rules for each deviation level

Level	**Radius**	**Avg. Num. Rules**	
		Seq. Niching	Det. 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

4.4.2.1 Results and discussion

Table 4.7 shows the number of rules generated by the genetic algorithm with the previous technique (NSDR with SN) and with the new DC scheme (NSDR with DC). The new technique produces less rules. This suggests the possibility that the new 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 exhibits that the new technique is able to find a set of more compact rules producing the same performance. This can be explained by the fact that sequential niching is more sensitive to the definition of the distance between individuals than deterministic crowding.

Another notable point is the efficiency of the DC technique. The DC technique only needs four runs (one per level) to generate a rule set. For the previous 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.

In Section 4.3.3, it is shown that the NSDR technique produces a good estimate of the level of deviation when this is calculated using D_∞ distance. Table 4.8 shows the confusion matrix for the NSDR technique using sequential niching and deterministic crowding. For each element in the testing set, the function $\mu_{non\_self}(\overrightarrow{x})$ 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 D_∞ distance). Each row (and column) corresponds to a range or level of deviation. The ranges are

Table 4.8: The values of the matrix elements correspond to the number of testing samples in each class. The diagonal values represent correct classification

PC output level	**NSDR output level**				
	Seq. niching				
	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
	Det. 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

specified on square brackets. A perfect output from the NSDR algorithm will generate values only in the diagonal.

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

4.5 Extending NSDR to use fuzzy rules

The purpose of this section is to extend the algorithm described in the previous section to evolve fuzzy rules instead of crisp rules. That is, given a set of self samples, the algorithm will generate fuzzy detection rules in the non-self space that can determine if a new sample is normal or abnormal. As it will be shown later, the use of fuzzy

rules improves the accuracy of the method and produces a measure of deviation from the normal that does not need to partition the non-self space.

4.5.1 Anomaly detection with fuzzy rules

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

$(x_1, \ldots x_n)$: element of the self/non-self 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 $\widehat{T_i} \subseteq S$ associated to each fuzzy set T_i,

$$T_i = \bigcup_{S_j \in \widehat{T_i}} S_j,$$

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

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

An example of fuzzy detection rules in the self/non-self 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

4.12 shows an example of such a division using five basic fuzzy sets representing the linguistic values *Low*, *Medium-Low*, *Medium*, *Medium-High* and *High*.

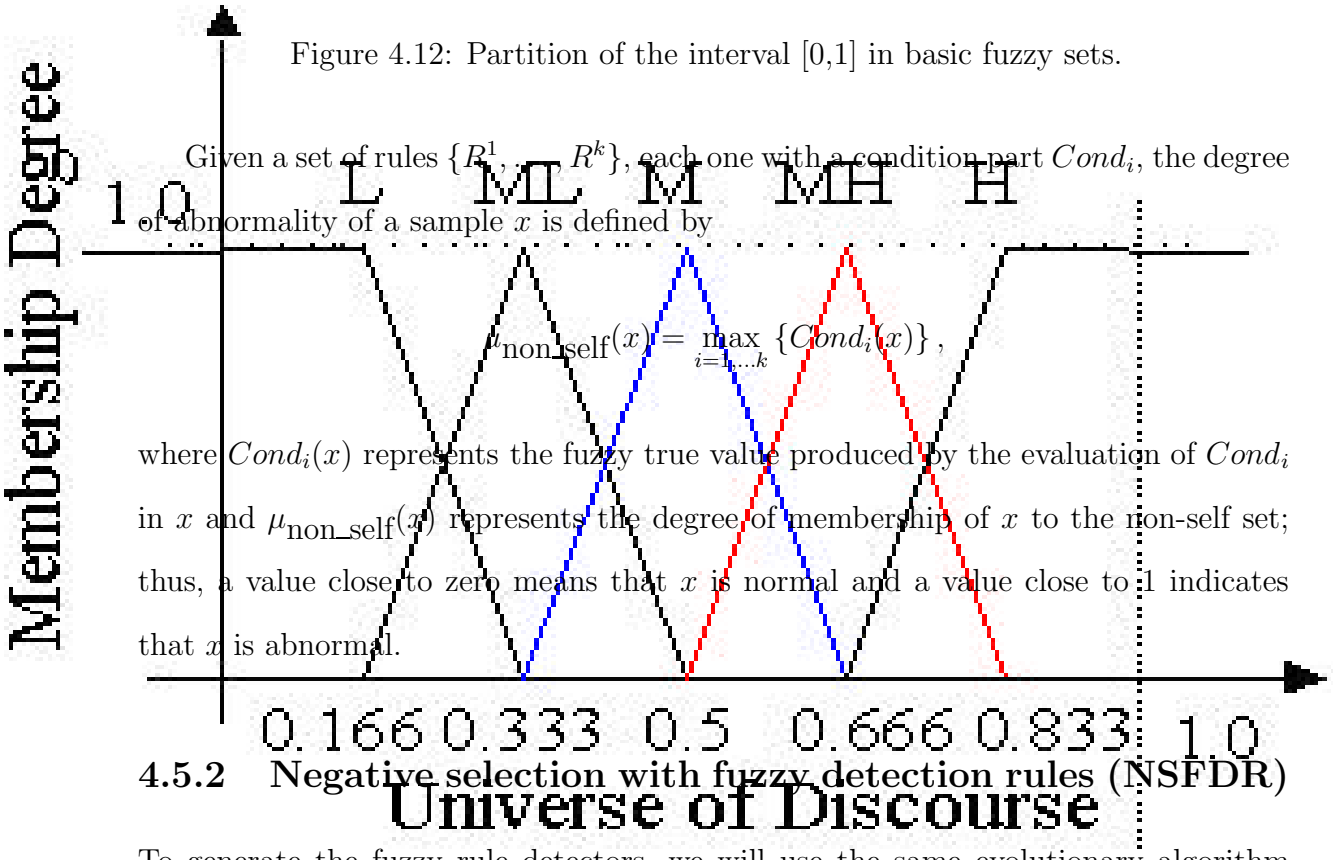


Figure 4.12: Partition of the interval [0,1] in basic fuzzy sets.

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 non-self set; thus, a value close to zero means that x is normal and a value close to 1 indicates that x is abnormal.

4.5.2 Negative selection with fuzzy detection rules (NSFDR)

To generate the fuzzy rule detectors, we will use the same evolutionary algorithm described in the Subsection 4.4.1 (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 deterministic crowding algorithm. Figure 4.13 shows the NSFDR algorithm. The time complexity of the algorithm is $O(num\_gen \cdot pop\_size \cdot |Self'|)$.

```
NS-Fuzzy-Detector-Rules(Self′)
   Self′  :  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 4.13: Negative selection with fuzzy detection rules (NSFDR) algorithm.

The use of fuzzy rules requires changes on the chromosome representation, fitness evaluation, and distance calculation. These changes are described below.

4.5.2.1 Chromosome representation

Each individual (chromosome) in the genetic algorithm represents the condition part of a rule, since the consequent part is same for all rules (the sample belongs to non-self). As it was described before, a condition is a conjunction of atomic conditions. Each atomic condition, $x_i \in T_i$, 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 4.14 shows the structure of a chromosome which is $n \times m$ bits long (n is the dimension of the space and m is the number of basic fuzzy sets).

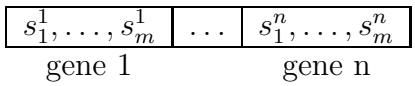


Figure 4.14: Structure of the chromosome representing the condition part of a rule. Each gene represents an atomic condition $x_i \in T_i$ and each bit s_j^i is 'on' if and only if the corresponding basic fuzzy set S_j is part of the composite fuzzy set T_j.

4.5.2.2 Fitness evaluation

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:

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

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

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

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

The fitness is defined as follows:

$$fitness(R) = C \cdot (1 - selfCovering(R)) + (1 - C) \cdot volume(R),$$

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

4.5.2.3 Individual's distance calculation

In this work, we use the 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 (see Figure 4.14) in both parent and child fuzzy rule detectors is set to one, 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.

4.6 NSFDR experimentation

We applied the fuzzy algorithm (Negative Selection with Fuzzy Detection Rules - **NSFDR**) and the crisp version (NSDR using DC) to three different data sets as shown in Table 4.9.

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 are combined to define the final set of detectors produced by the NSDR.

Table 4.9: Data sets used for experimentation

Data Set	**Training**	**Testing**	
		Normal	**Abnormal**
Mackey-Glass	497	396	101
MIT-Darpa 99	4000	5136	56
MIT-Darpa 98	1474	19056	396745

In order to asses the performance of both methods, we calculate the detection rate (**DR**, Equation 4.1) and false alarm rate (**FA**, Equation 4.2) and plot the result using ROC curves (as described in Subsection 4.3.2.1 on page 68). Also, the reported DR was obtained for each algorithm when the FA was fixed to 3%.

4.6.1 Experiments with Mackey-Glass time series

4.6.1.1 Data set and preprocessing

The Mackey-Glass series [109] has been used as a test set for different anomaly detection approaches [110, 15, 111]. Although it is generated by a deterministic differential equation, it exhibits a chaotic behavior that makes its prediction difficult.

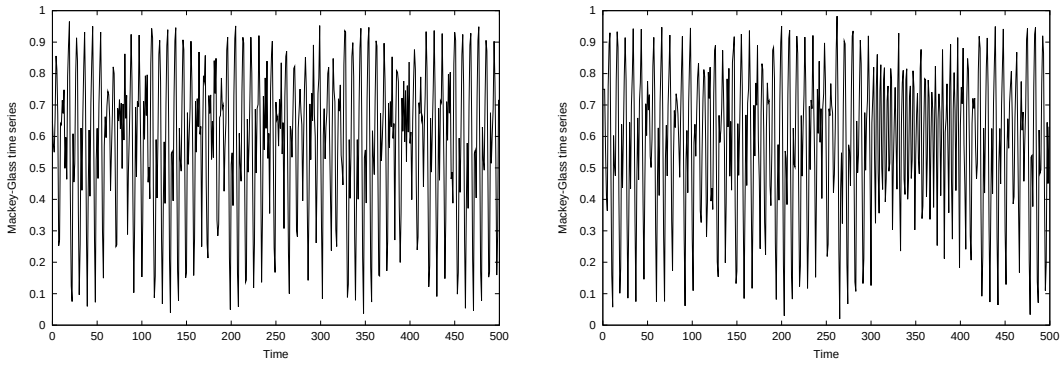
The differential equation that defines the series is the following:

$$\frac{dx}{dt} = \frac{ax(t-\tau)}{1+x^c(t-\tau)} - bx(t) \tag{4.3}$$

The parameter τ controls the complexity of the series dynamics, ranging from periodic to chaotic behavior.

In order to generate the training and testing data sets, Equation 4.3 is solved numerically using the fourth-order Runge-Kutta method with an integration step of 0.02, a sampling rate of 12, and an initial value vector with all its elements equal to 1.1. The parameter values used for the equation are: $a = 0.2$, $b = 0.1$, and $c = 10$, which are the general choice in the literature [15, 110].

The normal samples were produced from a time series with 500 elements generated using $\tau = 30$ and discarding the first 1000 samples to eliminate the initial value effect. The resulting time series is shown in Figure 4.15(a).



(a) (b)

Figure 4.15: Mackey-Glass time series. (a) normal, using $\tau = 30$; (b) with an anomaly, $\tau = 17$ from 300 to 400.

The test data (Figure 4.15(b)) is generated as before using $\tau = 30$, but starting with different initial conditions. An abnormality is introduced between time 300 to 400 by changing the parameter τ to 17. It is important to note that this experimental setting is different from the one used by Dasgupta and Forrest [15]. In that work, the anomalous time series is identical to the normal one, with the exception of the portion between 1000 and 1500. In our case, the two series are completely different since they were generated using different initial conditions. This makes the problem more challenging for the anomaly detection algorithm, since it has to be able to learn the structure of the normal set and not just memorize the samples.

The features are extracted using a sliding overlapping window of size n. If the time series has the values: $x_1, x_2, ..., x_m$, the feature set generated from it will be the following:

$$\begin{array}{llll} (x_1, & x_2, & ... & x_n) \\ (x_2, & x_3, & ... & x_{n+1}) \\ \vdots & \vdots & \vdots & \vdots \\ (x_{m-n+1} & x_{m-n+2} & ... & x_m) \end{array}$$

So, from a time series with m elements and using a sliding window of size n, we can generate (m-n+1) samples. In our experiments, we used $n = 4$. All the vector components are normalized to the interval [0,1].

4.6.1.2 Results and analysis

The proposed approach (NSFDR) performs better than the crisp algorithm (NSDR), see Figure 4.16. This behavior can be attributed to the fuzzyfication of the search space, because the fuzzy rule detectors provide a better characterization of the normal-abnormal boundaries.

Table 4.10 compares the performance of the tested algorithms over the Mackey-Glass data set. When the FA rate is fixed to 3%, the NSFDR algorithm is able to

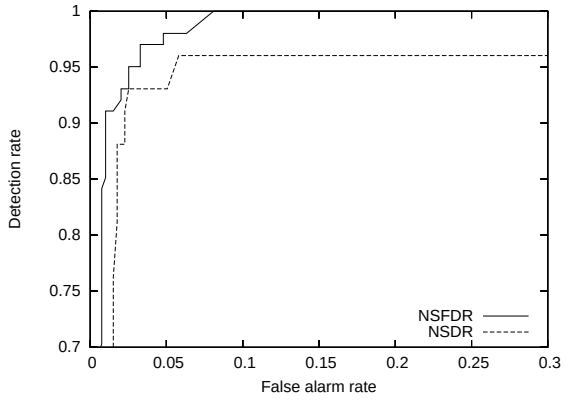


Figure 4.16: ROC curves generated by the two algorithms tested with the Mackey Glass data set.

Table 4.10: Comparative Performance in the Mackey-Glass Problem

Algorithm	**DR%**	**# Detectors**
NSFDR	**95.05**	**14**
NSDR	93.07	78

detect a higher percentage of abnormal samples than the NSDR algorithm. However, the main advantage of the fuzzy method seems to be the smaller number of detectors that it generates. This suggest that the fuzzyfication of the search space allows a simple characterization of the abnormal (non-self) space.

4.6.2 Experiments with network traffic data

We used the MIT-Darpa 99 data set described in Subsection 4.3.1. Additionally, we used the data set corresponding to the 1998 version of the DARPA intrusion detection evaluation also prepared and managed by MIT Lincoln Labs [100]. The data set 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 at the University of California Machine Learning

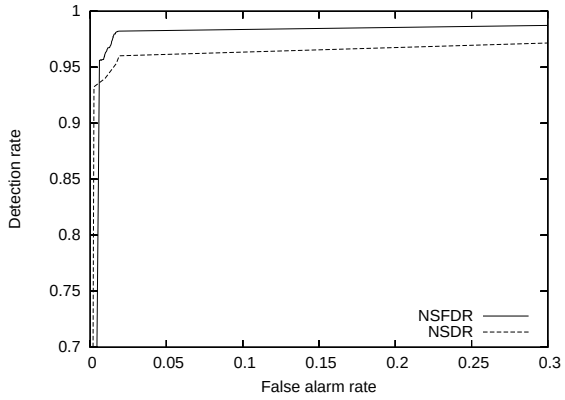


Figure 4.17: ROC curves generated by the two algorithms tested with the MIT-Darpa 98 data set.

repository[4] [112]. Even though the data set corresponds to a 10% of the original data, its size is still considerably big (492,021 records).

We generated a reduced version of the 10% data set including only the numerical attributes. Therefore, the reduced 10% data set is composed by 33 attributes. The attributes were normalized between 0 and 1 using the maximum and minimum values found. Of the normal samples, 80% were picked randomly and used as training data set, while the remaining 20% 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 data set (randomly generated) was used as a training data set. Henceforth we will call this data set **MIT-Darpa 98**.

4.6.2.1 Results and analysis (MIT-Darpa 98)

The NSFDR algorithm shows a better performance than the NSDR algorithm (Figure 4.17). The results of the NSDR algorithm are competitive only for a high FA rate (greater than 4%). Table 4.11 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 4.11 is the closest value to the optimal point (0,1). Amazingly, the number of detectors using fuzzyfication is very small compared to the number of

[4] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

Table 4.11: Comparative performance in the MIT-Darpa 98 data set

Algorithm	DR%	FA%	# Detectors
NSFDR	**98.22**	**1.9**	14
NSDR	96.02	1.9	699
EFRID[113]	98.95	7.0	-
RIPPER-AA[114]	94.26	2.02	-

detectors using the crisp characterization. This suggest that the fuzzy representation could handle high dimensionality better (the dimensionality of this data set is 33 attributes).

According to Table 4.11, the performance of NSFDR is comparable with the performance of approaches reported in the literature and in many cases performs better. For example, when NSFDR is compared with RIPPER-AA, the FA rate is almost the same (close to 2%), but NSFDR has a higher DR (4% more abnormal samples detected). Now, compared with the crisp approach (NSDR) the performance is also superior (2.2% more abnormal samples detected). Clearly, the fuzzy characterization of the abnormal space reduces the number of false alarms while the detection rate is increased.

4.6.2.2 Results and analysis (MIT-Darpa 99)

The performance of the NSDR algorithm is better than the performance of NSFDR algorithm for very small values of the FA rate. However, if the FA rate is allowed to be at most 2%, the NSFDR is clearly superior (Figure 4.18).

Table 4.12 compares the performance of the tested algorithms over the MIT-Darpa 99 data set (for a FA rate less than 3%). Again, the fuzzy method (NSFDR) generates a smaller set of rules without sacrificing the performance. This supports our claim that the fuzzy representation permits a more compact representation of the self/non-self space.

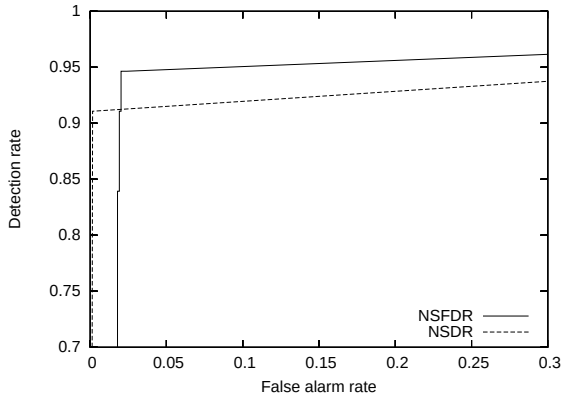


Figure 4.18: ROC curves generated by the two algorithms tested with the MIT-Darpa 99 data set.

Table 4.12: Comparative Performance in the MIT-Darpa 99 Problem

Algorithm	**DR%**	**# Detectors**
NSFDR	**94.63**	**7**
NSDR	89.37	35

4.7 Summary

In this chapter, we investigate a technique to perform anomaly detection based on the negative selection algorithm (NSDR). This technique uses a genetic algorithm to generate good anomaly detectors rules. In order to test this technique, a set of experiments to detect anomalies in network traffic data were performed. We used a real world data set (MIT-Lincoln lab), used by different researchers in computer security, for testing. The following are some preliminary observations:

- The immuno-genetic 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 negative selection algorithm 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 positive characterization (with a detection rate of 87.5% and a maximum false alarms rate of 1%), while only using a fraction of the space (when compared to positive characterization).

- The use of deterministic crowding as a niching technique improved the results obtained using sequential niching. While keeping 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 since it is able to generate multiple rules for each individual run of the GA.

In Subsection 4.5, the NSDR technique was extended to generate fuzzy rules. 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 new approach:

- It provides a better definition of the boundary between normal and abnormal. The previous approach used a discrete division of the non-self space, whereas the new approach does not need such a division since the fuzzy character of the rules provide a natural estimate of the amount of deviation from the normal.

- It shows an improved accuracy on the anomaly detection problem. This can be attributed to the fuzzy representation of the rules that reduces the search space, allowing the evolutionary algorithm to find better solutions.

- It generates a more compact representation of the non-self space by reducing the number of detectors. This is also a consequence of the expressiveness of the fuzzy rules.

Chapter 5

A Hybrid Immune Learning Algorithm for Anomaly Detection

5.1 Introduction

In the previous chapter, we presented an approach that allows the extraction of high-level knowledge expressed in the form of *If-Then* rules (crisp and fuzzy) by using an evolutionary NS algorithm. The purpose of this chapter is also to exploit the regularities of the self/non-self space to produce a high-level anomaly detection function. The new approach also works in a self/non-self space represented as a subset of $\mathbb{R}^n$. However, this approach is quite different from the previous one:

- The detectors are represented as points in $[0, 1]^n$. They do not have additional structure and there is not a high-level interpretation for them (unlike the previous approach where the detectors where interpreted as rules).
- The detectors are generated by a heuristic algorithm based on negative selection (not evolutionary), which tries to sparse the detectors in the non-self space.
- The detectors are not used directly to detect anomalous elements; instead, a high-level anomaly detection function is generated by a hybrid immune system

that combines the detector generator algorithm with a conventional classification algorithm.

In Section 5.2, we discuss the application of conventional machine learning techniques to solve the anomaly detection problem. Also, this section discusses the advantages and disadvantages of negative detection approach. This discussion will provide the context for a hybrid immune learning algorithm for anomaly detection [26, 24] (in Section 5.3) that combines the NS algorithm and a classification technique. The hybrid immune-based approach uses a real-valued negative selection algorithm (**RNS**) [26, 24], as presented in Section 5.4. Section 5.5 illustrates the hybrid approach to extract high-level knowledge expressed in terms of fuzzy classifier rules [24]. In Section 5.6, the proposed technique is compared against other anomaly detection methods [25, 26] including one based on self-organizing maps (explained in Subsection 5.6.1).

5.2 Anomaly detection and learning

The anomaly 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 (see Section 2.3). 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. The next subsection discusses some approaches to solve the anomaly detection problem from a machine learning perspective.

5.2.1 Anomaly detection problem from a machine learning perspective

In general, a two-class classification method uses a set of samples (from both classes) to build a model that can discriminate between the two classes. Usually, the model defines a boundary between the classes. But, what happen if samples only from the

normal class are available for training? It would not be clear as to where to put the decision boundary. For instance, a model that classifies everything as normal will be according to the training samples, but it is clearly useless. A better approach will look for a more specific model that can fit the training samples.

Notice that if no assumption about the shape/structure of the self set is made, the most specific model is the set of self samples itself. That is, a new sample is classified as normal only if the sample is in the training set. This is not useful for many applications; so, it is necessary to make an assumption (inductive bias) about the type of model of normal (self) that we want to induce[1].

The problem, then, is reduced to find a model that fits the training samples and is consistent with the inductive bias. The *candidate-elimination* algorithm [116] can be used to find it. The basic idea of the algorithm is to represent the set of feasible hypotheses (set of consistent models, also called *version space*) using two limits: the *general boundary* (set of more general hypotheses) and the *specific boundary* (set of more specific hypotheses). A positive (normal) sample will change the specific boundary to make it more general and a negative sample will change the general boundary set to make it more specific. The algorithm can be applied even if no negative samples are available. In this case, the general boundary set is not changed. But, this is not a problem since we are interested in the most specific model. The main drawback of this approach is that it is only feasible if the space of possible models is simple, e.g. small number of discrete features.

Another possibility is not to assume a model at all. Instance based classification methods, like nearest neighbor classification [101], use the training samples to perform classifications; thus, they do not need to assume a model. A nearest neighbor classifier assigns the class to a new sample depending on the classes of its nearest neighbors in the training set. If we apply this algorithm to a training set composed only of normal samples, all the new samples will be classified as normal. However, it is possible to

[1]Notice that the inductive bias is necessary, since, as it was pointed out by Mitchell [115], "*a learner that makes no a priori assumptions regarding the identity of the target concept has not a rational basis for classifying any unseen instances.*"

extend this algorithm to use only normal samples: a new sample is classified as normal or abnormal depending on the distance to the nearest neighbor in the training set; if the distance exceeds a given threshold, the sample is classified as normal, otherwise it is abnormal. This method is similar to the one described in Subsection 4.3.2.

The inductive bias of this method is: *if an element is close to a normal sample, it is highly probable that it is normal*. The main drawback of this method is that it is necessary to store all the training samples; the cost of this can be prohibitive for many practical applications. This problem can be solved if the set of training samples is represented in a more compact way. For instance, a clustering method can be applied and then the cluster information can be used instead of normal samples to perform the classification task. A method based on this approach is described in Subsection 5.6.1.

5.2.2 Positive or negative detection?

The approaches described in the previous subsection build models of the normal set (positive detection). It is also possible to accomplish the same goal by building models of the abnormal set (negative detection), as in the NS algorithm (Subsection 2.2.1). Negative detection does not seem to be as natural as positive detection in cases where the normal is relatively small. So, what is the justification to do negative detection? Esponda and Forrest [117] provided three main reasons:

- There is practical evidence that the negative detection approach works since it has been applied with some success to solve practical problems.
- From an information theory point of view, to characterize the normal space is equivalent to characterize the abnormal space.
- Negative detection is more suitable for distributed anomaly detection. That is, it is possible to divide a set of negative detectors in subsets an apply them in a distributed fashion, since 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 one. However, if the description of the normal set is compact enough, it would be more efficient to have multiple, redundant copies of the positive detectors to 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.

Keogh et al. [111] argued that *"a major limitation of the approach* (negative selection) *is that is is only defined when the space of self is not exhaustive."* The authors provided an example of random walk data series, where the self set can have all possible patterns, causing the non-self 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, rather 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.

5.3 Proposed hybrid immune learning approach

The NS algorithm has been used mainly to perform negative detection, i.e. the detectors generated by the algorithm are used directly to identify elements in the abnormal (non-self) space. As it was discussed in the previous section, this approach is useful in some specific applications (distributed anomaly detection and a not very small normal subspace).

In this section, we propose a different use of the NS algorithm to perform anomaly detection. This approach uses neither negative nor positive detection; rather, the

approach tries to find the boundary (crisp or fuzzy) between normal and abnormal classes. This approach can be useful even if we are not performing distributed anomaly detection or when the normal set is small.

The basic idea is to use the RNS algorithm (to be presented in the next section) to generate non-self samples. Then, apply a classification algorithm to find a characteristic function of the self (or non-self). This characteristic function corresponds to the anomaly detection function (see Section 2.3.1).

Figure 5.1 illustrates the basic building blocks of the approach. During the training stage, the input corresponds to the normal samples (feature vectors) that are used by the RNS algorithm [24] to generate abnormal samples. Subsequently, the normal and abnormal samples are used as input to a supervised algorithm that produces a classifier. This classifier corresponds to the anomaly detection function and is used during the testing phase to classify new samples as normal or abnormal.

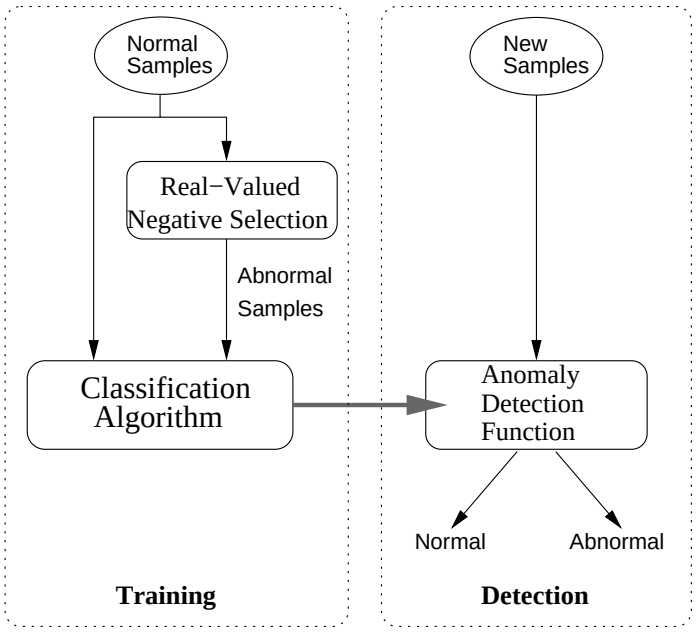


Figure 5.1: A hybrid immune system for anomaly detection that generates an anomaly characterization function from normal samples.

It is important to highlight that this technique allows the use of a supervised algorithm for a task that traditionally requires an unsupervised method (as it was discussed in Subsection 5.2.1). The main advantages of this approach are:

- The classification problem has been studied for a long time. There are different efficient algorithms that have been extensively tested and applied to solve problems in different fields.

- The approach does not require the modification of the classification algorithm. It allows a modular composition that makes easier to use widely available and well tested existing implementations of supervised algorithms.

- The classification problem is closer to the problem of anomaly detection than unsupervised learning problems, like clustering. Clustering methods group the input data based on the principle of maximizing the intraclass similarity and minimizing the interclass similarity. On the other hand, the main objective that drives a classification algorithm is to improve the accuracy of the classifier, that is, to improve the ability to distinguish between classes. This is clearly more related with the anomaly detection goal of maximizing the detection rate while keeping a low false alarm rate.

- It is possible to use real abnormal samples, if available, by combining them with the ones generated by the RNS algorithm and feeding them together to the classification algorithm.

The inductive bias of this method is the same as the inductive bias of the nearest neighbor method described in Subsection 5.2.1: an element close to a normal sample is likely to be normal. This assumption is reasonable for many applications, if the distance metric is well chosen.

There is no specific restriction on the kind of classification algorithm that can be used. For instance, in Section 5.5, we use a fuzzy rule evolver system, and in Section 5.6, a neural network based classifier is employed.

The abnormal samples generated by the RNS algorithm can be thought of as *artificial anomalies* [24]. The idea of generating artificial anomalies was also proposed by Fan et al. [114] independently. Unlike our work, the method proposed by Fan et al.

is addressed to generate artificial anomalies by perturbing real data samples (normal and abnormal). The perturbation is performed by randomly choosing a feature of a given sample and assigning a random value. The generated artificial anomalies are combined with the real data and fed to a learning algorithm. The artificial-anomaly generation method assumes that each feature takes a discrete set of values; hence, the algorithm cannot be directly applied to real-valued data. Moreover, the approach appears to be ad-hoc, whereas NS algorithm provides a systematic way of generating such anomaly detectors.

5.4 Real-valued negative selection (RNS)

As in the previous chapter, the self/non-self space, U, corresponds to a subset of $\mathbb{R}^n$. A detector (antibody) is defined by an n-dimensional vector that corresponds to the center and by a real value that represents its radius; therefore, a detector can be seen as a hypersphere in $\mathbb{R}^n$. The detector-antigen matching rule is expressed by the membership function of the detector, which is a function of the detector-antigen Euclidean distance and the radius of the detector (see Equation 5.1).

The input to the algorithm is a set of self samples represented by n-dimensional points (vectors). The algorithm tries to evolve another set of points (called antibodies or detectors) that cover the non-self space. This is accomplished by an iterative process that updates the position of the detector driven by two goals[2]:

- Move the detector away from self points.
- Keep the detectors separated in order to maximize the covering of non-self space.

The logical steps of the algorithm are shown in Figure 5.2, which are described in a more detailed way in Figure 5.3.

The parameter r specifies the radius of detection of each detector. Accordingly, for an antigen a to be detected by a detector d, the distance between d and a should

[2]This approach is similar to the NS greedy algorithm [44], but in a real-valued space.

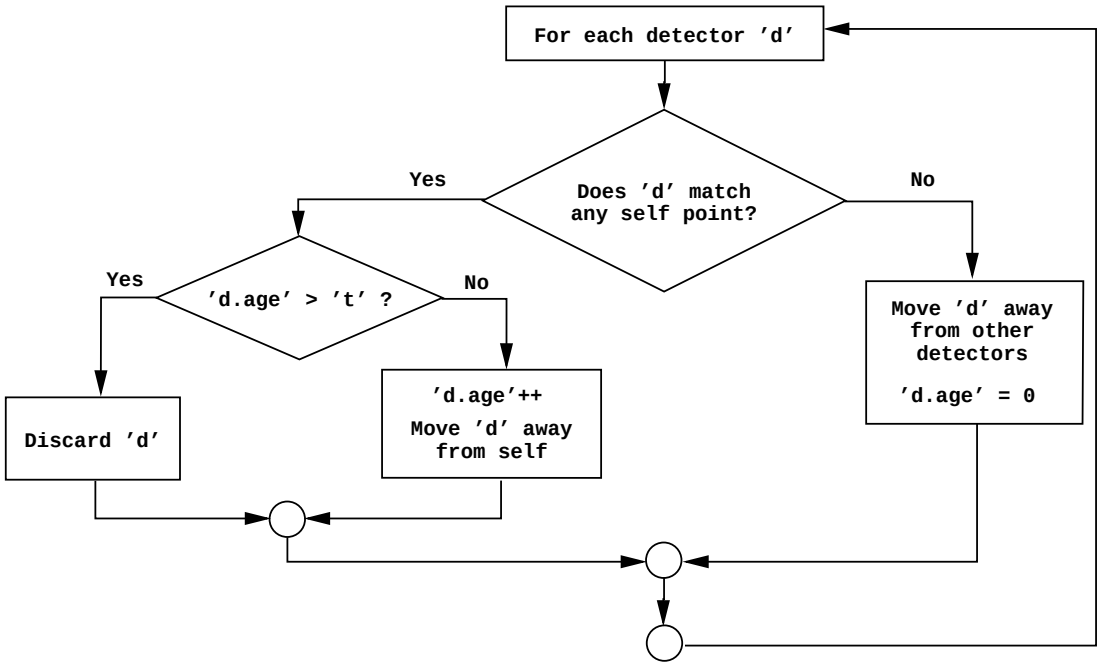


Figure 5.2: An illustration of an iteration of the real-valued negative selection algorithm.

be at most r. Since we do not want the detectors to match self points, the shortest allowable distance for a good detector to the self set is r. Therefore, the parameter r also specifies the allowed variability in the self space.

In order to determine if a detector d matches a self point, the algorithm calculates the k-nearest neighbors of d in the self set. It then calculates the median distance of these k-neighbors. If this distance is less than r, the detector d is considered to match self. This strategy makes the algorithm more robust to noise and outliers.

The function $\mu_d(x)$ is the matching membership function of the detector d. It indicates the degree of matching between x, an element of the self/non-self space, and d. It is defined as:

$$\mu_d(x) = e^{-\frac{\|d-x\|^2}{2r^2}} \tag{5.1}$$

Each detector has an assigned age that is increased at each iteration, if it is inside the self set. If the detector becomes old, i.e. it reaches the maturity age t and has not been able to move out of the self space, it will be replaced by a new randomly

```
REAL-VALUED-NEGATIVE-SELECTION(r,η,t,k)
   r :  radius of detection
   η :  adaptation rate, i.e., the rate at which the detectors
        will adapt on each step
   t :  once a detector reaches this age it will be considered
        to be mature
   k :  number of neighbors to take into account
 1: While stopping criteria is not satisfied
 2:  For each detector d do
 3:   NearCells ←k-nearest neighbors of d in the Self set
 4:   NearCells is ordered with respect to the distance to d
 5:   NearestSelf ← median of NearCells
 6:   If dist(d, NearestSelf) < r Then
 7:     dir ← (Σ_{c∈NearCells}(d−c)) / |Σ_{c∈NearCells}(d−c)|
 8:    If age of d > t Then     ▷ detector is old
 9:     Replace d by a new random detector
10:    Else
11:     Increase age of d
12:     d ← d + η · dir
13:    EndIf
14:   Else
15:    age of d ← 0
16:    dir = (Σ_{d'∈Detectors} μ_d(d')(d−d')) / |Σ_{d'∈Detectors} μ_d(d')(d−d')|
17:    d = d + η · dir
18:   EndIf
19:  EndFor
20: EndWhile
```

Figure 5.3: Real-valued negative selection (RNS) algorithm.

generated detector. The age is reset to zero when the detector is outside of the self space.

The parameter η represents the size of the step used to move the detectors. In order to guarantee that the algorithm will converge to a stable state, it is necessary to decrease this parameter in each iteration in such a way that $\lim_{i\to\infty} \eta_i = 0$. We use the following updating rule

$$\eta_i \leftarrow \eta_o e^{\frac{-i}{\tau}},$$

where η_0 is the initial value of the adaptation rate, and τ is a parameter that controls its decay.

The stopping criteria is based on a pre-specified number of iterations, *num_iter*. This produces a time complexity of $O(num\_iter \cdot num_{ab} \cdot (num_{ab} + |S'|))$, where num_{ab} is the number of detectors and $|S'|$ is the number of self samples.

5.5 Applying the hybrid immune learning approach to extract high level knowledge

The purpose of this section is to illustrate the application of the hybrid immune system approach (presented in Section 5.3) to the extraction of high level knowledge from a set of normal samples. The high level knowledge is expressed in terms of fuzzy rules.

For this experiment, we work with a classical data set used extensively in the pattern recognition literature, the Iris data set[3] [118]. In this set, there are three different classes of flowers: setosa, virginica and versicolor. Each element in the data set is described by four attributes.

The classification algorithm used is an evolutionary algorithm to generate fuzzy classifier rules [96]. This algorithm uses a genetic algorithm with a linear representation of tree structures in order to evolve complex fuzzy rule sets.

[3]This particular version of the data set was obtained from the University of California Machine Learning repository [112] at `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris`.

Three different experiments were performed in each case using one class as normal and the other two as abnormal. The training set was only composed by elements of the normal class, and no elements from the other classes were included.

The parameters used by the RNS algorithm were $r = 0.1$, $\eta = 1.0$, $t = 10$, and $k = 5$. The number of detectors was 200 and the number of iterations was 200. The fuzzy classifier rule evolution algorithm was run using a population of 200 individuals during 100 iterations.

The following is an example of a rule produced by the algorithm when the class virginica was considered as normal:

$$\textbf{If } (x_2 \in M \vee x_1 \notin S) \wedge (X_3 \in M \wedge x_1 \notin ML \wedge x_1 \notin L) \textbf{ Then } \textit{Normal}$$

An important characteristic of these kinds of rules is the comprehensibility. This allows extraction of useful high-level knowledge. These results are comparable to the ones produced by the same algorithm using training samples from all the classes [96]. The main difference is that in the present work we only used examples from one class in the training phase, which introduces more complexity to the problem.

As it was explained previously, the condition part of the rule is used as the anomaly detection function μ_{self} (see Subsection 2.3.1). In order to quantify the anomaly detection capability of the evolved functions, we used the crisp (t-cuts) version $\mu_{self,t}$. This allows us to determine the accuracy of the anomaly detection. The accuracy is given in terms of the detection rate (Equation 4.1) and false alarm rate (Equation 4.2).

Figure 5.4 shows the increase of the detection rate for the anomaly detection function evolved $\mu_{self,t}$, when the threshold t is increased. When the threshold is equal to zero, everything is reported as normal. When the value is increased, the number of detections increases; however, the number of false alarms also increases.

Figure 5.5 shows the trade-off between detection rate and false alarm rate using ROC curves (as described in Subsection 4.3.2.1 on page 68). In all the cases, high

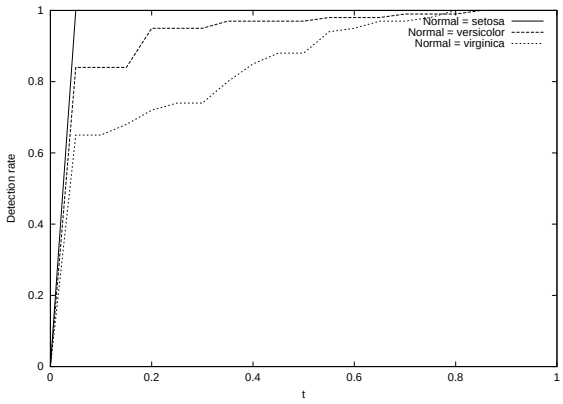


Figure 5.4: Evolution of the detection rate of $\mu_{self,t}$ when the threshold t is varied from 0 to 1.

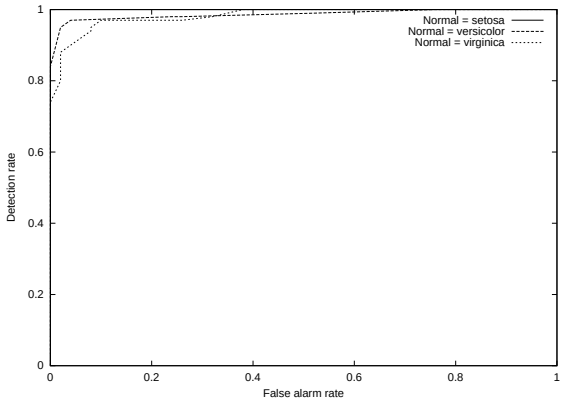


Figure 5.5: ROC curve for the evolved anomaly detection function $\mu_{self,t}$ (the curves corresponding to setosa and virginica overlap).

detection percentages were reached even for small false alarm rates. Some illustrative results are shown in Table 5.1.

5.6 Comparing the hybrid immune learning approach with other anomaly detection techniques

In order to test the proposed hybrid immune anomaly detection technique, we applied it to four different data sets. Each data set is divided in two subsets: the training data set, which contains only normal samples, and the test data set, which contains

Table 5.1: Accuracy of the evolved anomaly detection function when a maximum false alarm rate of 2% is allowed. (TP=True Positives, FP=False Positives, TN=True Negatives, FN=False Negatives)

Normal Class	**TP**	**FP**	**TN**	**FN**	**Detection Rate**
Setosa	100	0	50	0	100%
Virginica	95	1	49	5	95%
Versicolor	88	1	49	12	88%

a mixture of normal and abnormal samples. We use a multi-layer perceptron (**MLP**) trained with back-propagation [119] as the classification algorithm. In the remaining part of this chapter, we refer this technique as Hybrid Neuro-Immune System (**HNIS**).

In order to compare the performance, we also apply binary negative selection (**BNS**) using the NS greedy algorithm (see Section 2.2.1) and an anomaly detection technique based on self-organizing maps (**SOM**) (to be explained in the next subsection) to the same data sets. These techniques are compared in terms of classification accuracy using ROC curves as described in page Subsection 4.3.2.1.

The sensitivity of the system is controlled by a threshold that determines when a new sample is normal or abnormal. By varying this threshold, we can obtain different values for the detection and false alarm rates which are used to plot ROC curves. In the case of HNIS and SOM techniques, whose output is a continuous value, the threshold is a value between 0 and 1.

The output of the BNS technique is not continuous, being just 0 (normal) or 1 (abnormal); so, it does not make sense to apply a threshold to the output. It is possible to use the parameter r, which defines the size of the matching window, as a threshold that we can vary to produce different points of the ROC curve. However, we have to be careful when interpreting the results, since, unlike the other two methods, each point of the ROC curve will correspond to a different set of detectors.

5.6.1 Anomaly detection using self-organizing maps

A self-organizing map (SOM) is a type of neural network that uses competitive learning [119, 120]. A SOM is able to capture the important features contained in the input space and provides a structural representation that preserves a topological structure. The output neurons of a SOM are organized in a one- or two-dimensional lattice. The weight vectors of these neurons represent prototypes of the input data that can be interpreted as the centroids of clusters of similar samples.

In our experiments, we used SOM to cluster the normal samples. After the network is trained, the generated clusters are used to determine if a new sample is normal or abnormal. The basic idea is: if a new sample is 'close' enough to a normal cluster, it is considered normal; otherwise, it is classified as abnormal.

In general, we have a distance function $dist(s, K)$ that measures how close the sample s is to the cluster, K. To determine the abnormality of a new sample, the following function is used:

$$\chi_{abnormal}(s) = \begin{cases} 1 & \text{if } dist(s, Normal) \geq t \\ 0 & \text{otherwise} \end{cases} , \tag{5.2}$$

where,

$$dist(s, Normal) = \min\{dist(s, K_i) \mid K_i \in C\} , \tag{5.3}$$

and C is the set of clusters (found by the SOM algorithm) that represents the normal sub-space.

If we think the function $dist(s, Normal)$ is a kind of membership function[4] of the abnormal subspace, the function $\chi_{abnormal}(s)$ corresponds to the crisp version of it. In this case, the value t represents a threshold that defines the boundary between the normal and abnormal classes (see Subsection 2.3.1).

[4]Strictly speaking, this is not a membership function since it is not bounded. However, we can apply, for instance, a sigmoid function to make it bounded.

In order to determine a good distance measure $dist(s, K)$, we tested three options (in all the cases w_K, neuron weights, represents the centroid of the cluster K):

- **Euclidean distance**. This is the natural (or naive) choice since the SOM algorithm uses it to determine if a sample belongs to a given cluster:

$$dist(s, K) = \|s - w_K\| \tag{5.4}$$

- **Normalized distance**. The idea is to take into account the size of the cluster. Some clusters can be very sparse and others can have all the elements concentrated around the centroid. A measure of the size is the standard deviation. So, the standard deviation of the distance to the centroid of all the elements in a cluster (σ_K) is calculated and it is used to normalize the distance:

$$dist(s, K) = \frac{\|s - w_K\|}{\sigma_K} \tag{5.5}$$

- D_∞ **Minkowsky distance**. The Euclidean distance gives the same importance to all the features. So, it is possible that a sample with a non-negligible deviation in one feature will be considered as having the same overall deviation as a pattern with small deviations on many features. The D_∞ distance only takes into account the maximum of the differences for all the features:

$$dist(s, K) = \max\{|s_i - w_{K_i}| \text{ for } i = 1, \ldots, n\} \tag{5.6}$$

5.6.2 Mackey-Glass time series experiments

We used the Mackey-Glass time series data set described in Subsection 4.6.1.1.

5.6.2.1 Experimental settings

For the HNIS technique, the RNS algorithm was run using as input the training data to generate 400 detectors. The parameter values for the algorithm were: $r = 0.1$, $\eta = 1$, $t = 5$, and $k = 1$. The number of iterations was set to 400. The MLP used had 4 inputs and 1 output. Three different architectures were tested with 6, 12, and 16 hidden neurons. The parameters of the back-propagation algorithm were: learning rate 0.05, momentum 0.9, and number of iterations 4000.

For the BNS algorithm, the data was converted to binary strings assigning 5 bits to each feature and using binary and gray coding. This produced binary strings of length 20. The value of r was varied from 6 to 12. The greedy algorithm was run setting the failure probability to 0.

Three different SOM topologies were used: 4 input nodes with an output layer of 4×3, 4×6 and 6×6 neurons, respectively. The weights were initialized using random vectors. The SOM training algorithm was run for 1000 iterations using a Gaussian neighborhood. The initial and final learning rate were 0.1 and 0.005 respectively. The initial σ value was 5 and the final was 0.2.

5.6.2.2 Results

Figure 5.6 shows a typical output of three techniques when applied to the testing set. An output value close to "0" means normalcy. Each figure corresponds to the best result found by each method. Despite the fact that in all cases the output shows an increased activity in the abnormal region, there are peaks in the normal region that do not allow to establish a clear boundary between normal and abnormal. In order to smooth the anomaly detection function, a moving average filter was applied. The new output $\widehat{O_t}$ is calculated from the old output O_t using the following formula:

$$\widehat{O_t} = \frac{\sum_{i=1}^{s} O_{t-i}}{s}, \tag{5.7}$$

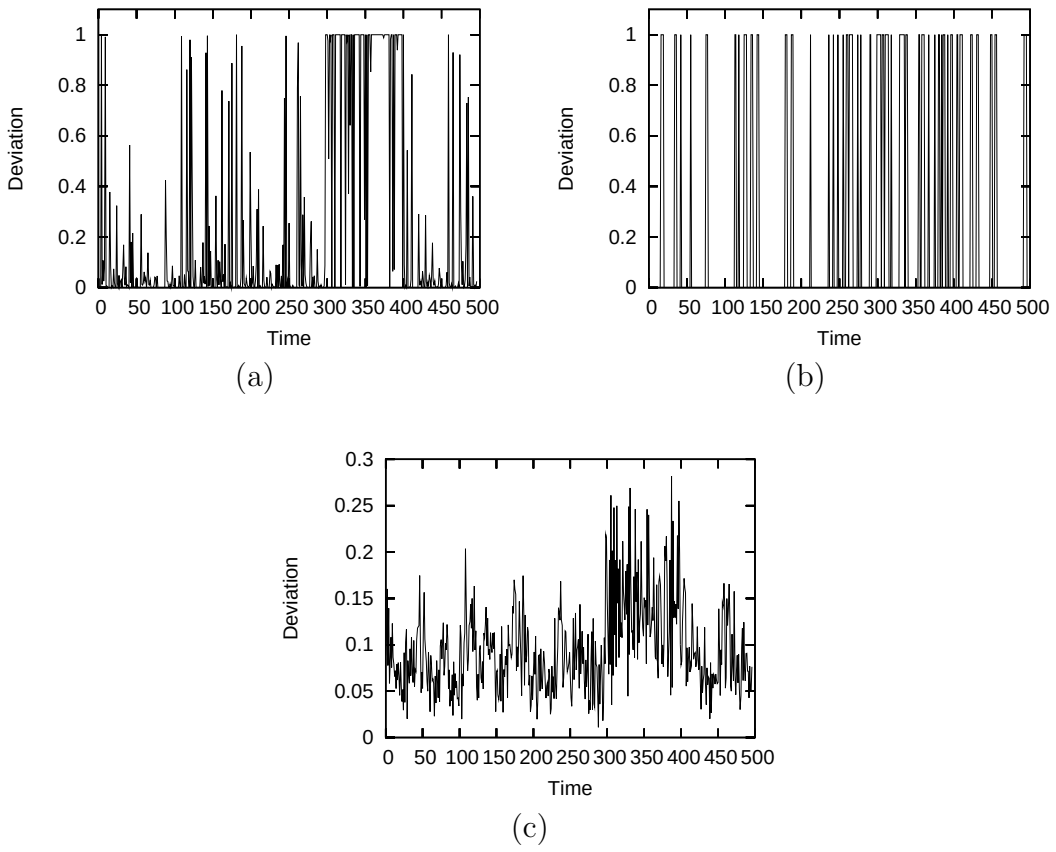


Figure 5.6: Output value produced by the anomaly function when applied to the Mackey-Glass testing set. (a) HNIS (12 hidden neurons). (b) BNS ($r = 8$, Gray coding). (C) SOM (6×6, D_∞ distance).

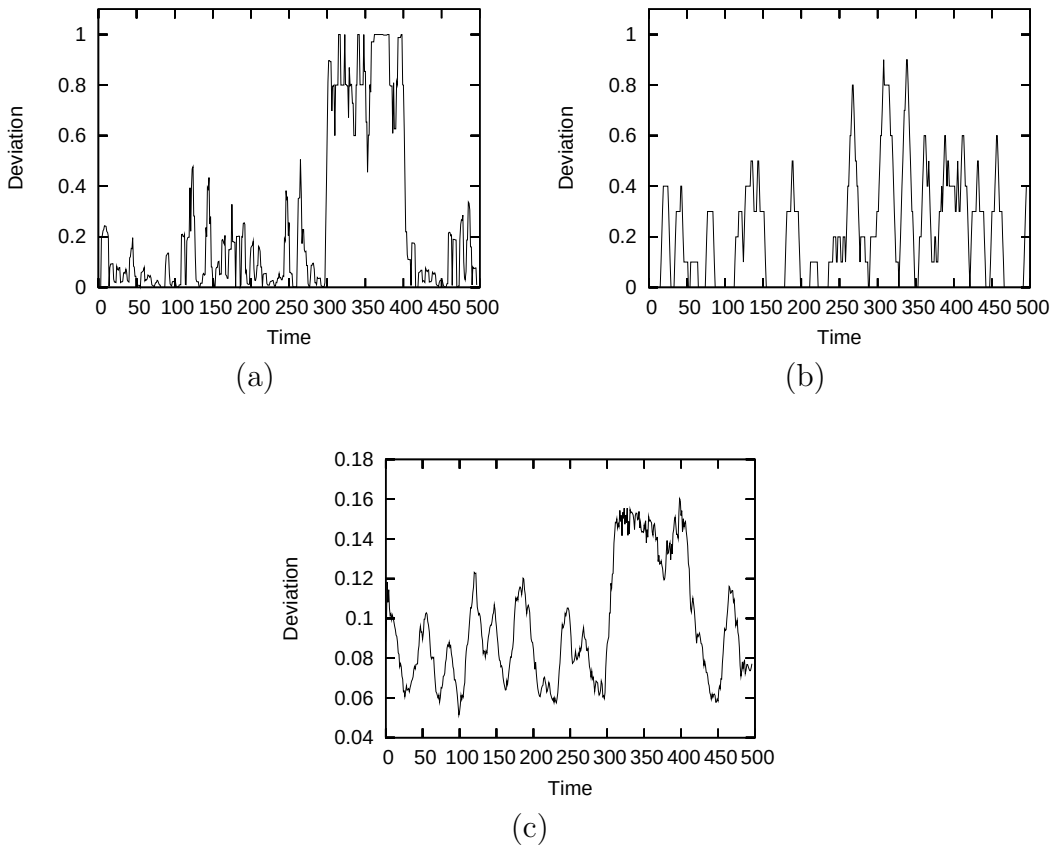


Figure 5.7: Output value, smoothed using Equation 5.7, produced by the anomaly function when applied to the Mackey-Glass testing set. (a) HNIS (12 hidden neurons, $s = 5$). (b) BNS ($r = 8$, Gray coding, $s = 10$). (C) SOM (6×6, , D_∞ distance, $s = 10$).

where s is the smoothing factor and indicates the size of the averaging window. The filter was applied to the output produced by each technique. Different values of s were tested, choosing the value that produced the best result for each individual technique. Figure 5.7 shows the smoothed versions of the outputs in Figure 5.6.

The following subsections shows more details of the results produced by each technique.

BNS results The number of detectors generated by the NS greedy algorithm are summarized in Table 5.2. These values coincide with the values predicted by the theoretical analysis described by D'haeseleer et al. [44].

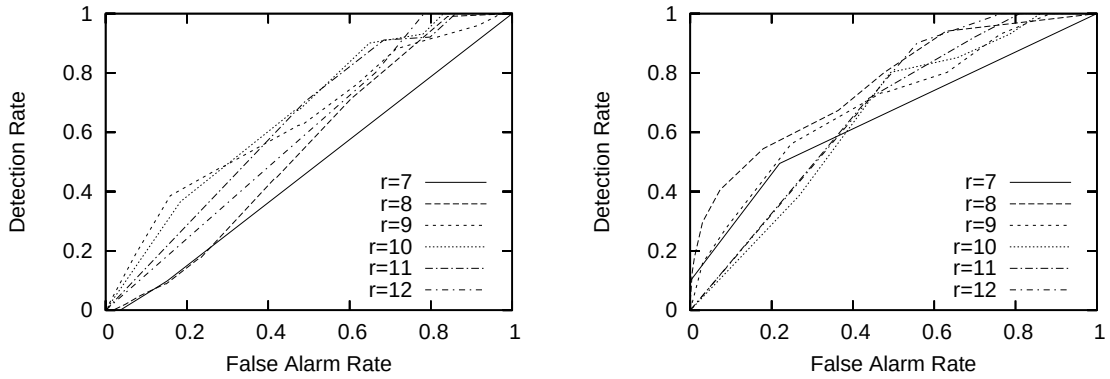
Table 5.2: Number of detectors produced by BNS (greedy) algorithm when applied to the Mackey-Glass training set

	Number of detectors	
r	**Binary encoding**	**Gray encoding**
6	0	0
7	13	19
8	90	79
9	300	301
10	736	750
11	1683	1705
12	3691	3709

The performance of the different set of detectors is shown in the ROC curves in Figure 5.8. It is important to note that it is possible to generate these ROC curves for each detector because of the smoothing process. This generates a continuous anomaly function that takes values between 0 and 1; so, it makes sense to use a threshold to decide when a given sample is normal or abnormal.

The results using Gray coding are in general better than the results produced with Binary coding. This is explained by the fact that Gray coding is more compatible with the kind of matching rule used by the BNS algorithm, r-contiguous matching. This is a fact that has been addressed by Dasgupta and Majumdar [121]. The best result is produced with a set of detectors generated using $r = 8$. An increase in r does not improve the performance, as it is shown by the ROC curves for $r = 9$ to $r = 12$, which are bound by the ROC curve generated with $r = 8$.

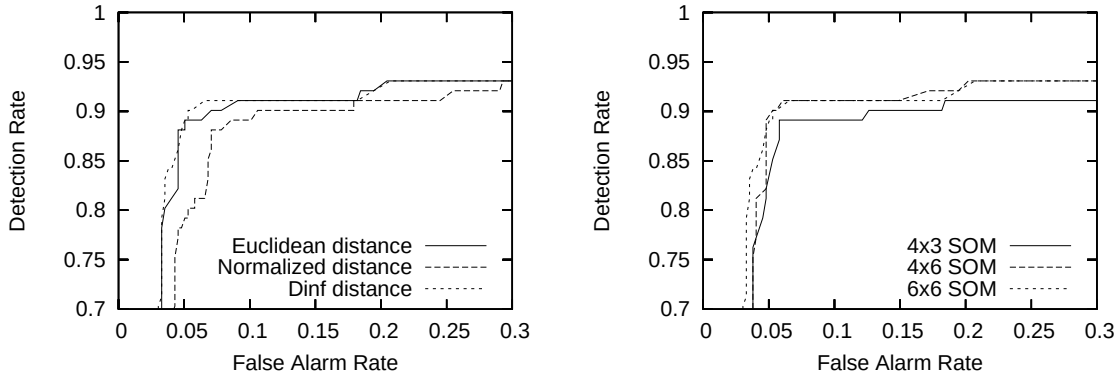
SOM results As it was discussed in Section 5.6.1, three different distance measures were proposed to calculate the anomaly detection function defined in Equation 5.2. Figure 5.9(a) shows the ROC curves corresponding to these distance measures. D_∞ Minkowsky distance (Equation 5.6) shows a slight advantage over other distance measures. Figure 5.9(b) shows ROC curves for different topologies of the SOM network. A higher number of neurons produces a most accurate classification; however, the



(a) Binary encoding (b) Gray encoding

Figure 5.8: ROC curves for BNS algorithm applied to Mackey-Glass test data set.

difference between the curves is not big; this suggests that a further increase in the network complexity may not improve the accuracy.



(a) (b)

Figure 5.9: ROC curves for SOM anomaly detection applied to Mackey-Glass test data set. (a) Different distance measures using 6×6 topology. (b) Different topologies using D_∞ distance.

HNIS results Figure 5.10 shows the ROC curves corresponding to different MLP topologies. The figure shows that an increase from 6 to 12 neurons improves the classification accuracy of the system. Accordingly, 12 neurons seem to be enough, since an increase to 16 does not produce any significant improvement in accuracy.

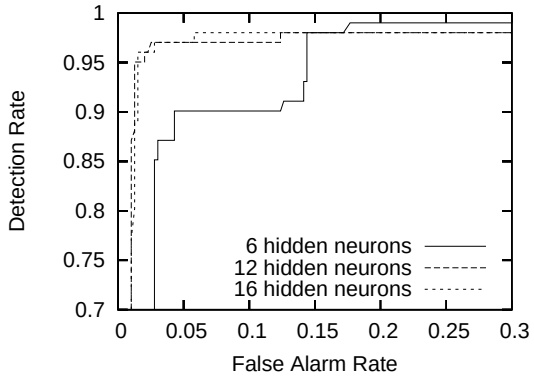


Figure 5.10: ROC curves for HNIS anomaly detection applied to Mackey-Glass test data set for different MLP topologies: 6, 12, and 16 hidden neurons.

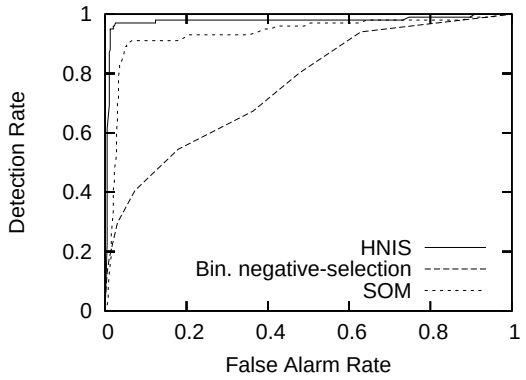


Figure 5.11: Best ROC curves produced by each method for the Mackey-Glass test data set.

5.6.2.3 Results comparison and discussion

The best performing configurations from each approach are compared in Figure 5.11. The configurations are: HNIS, 12 hidden neurons; BNS, Gray coding and $r = 8$; and SOM, 6×6 output layer and D_∞ distance. Clearly, HNIS has a better performance than other two methods. This shows that, at least for this specific data set, the combination of RNS with a MLP is able to capture the structure of the normal space, producing an anomaly detection function that can discriminate the normal and the abnormal new samples.

In contrast to the results reported by Dasgupta and Forrest [11], the performance of the BNS algorithm is very poor. This may be because of the experimental settings

used in our current work; the normal samples in the test data set are different from to those presented during training. This indicates that the anomaly detection algorithm should be able to generalize the structure of the normal set based on a limited subset of samples. Our hypothesis is that the binary (low-level) representation along with the r-contiguous matching rule (used by BNS) may not capture the high-level structure of the problem space.

5.6.3 Network traffic data experiments

We used MIT-Darpa 98 and MIT-Darpa 99 data sets as described in Subsections 4.6.2 and 4.3.1, respectively.

5.6.3.1 Experimental settings

The experimental settings for all the techniques are the same as the ones described in Section 5.6.2.1. The only differences are: for the MIT-Darpa 98 data set, the HNIS used 1000 detectors instead of 400, and for the MIT-Darpa 99 data set, three different MLP topologies were tested with 5, 9, and 18 hidden neurons, respectively.

5.6.3.2 Results comparison and discussion (MIT-Darpa 98)

The BNS algorithm was not able to generate a good set of detectors. We ran it for different values of r ranging from 6 to 12. The algorithm did not produce detectors for values of r less or equal to 8; however, for $r = 9$ the algorithm produced more than 2×10^8 detectors before it had to be manually stopped. This happened even with a failure probability as high as 0.5. Our hypothesis is that the high dimensionality of the space along with the small variability of the normal set makes it very difficult to cover the non-self space using the greedy algorithm. This result is similar to the one reported by Kim and Bentley [92].

Figure 5.12 shows the best ROC curves produced by the HNIS and SOM anomaly detection techniques. The configurations are: HNIS, 12 hidden neurons and SOM,

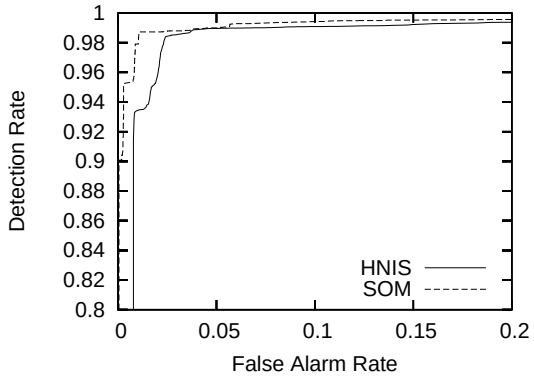


Figure 5.12: Best ROC curves produced by HNIS and SOM methods for the MIT-Darpa 98 test data set.

4×6 using D_∞ distance. The performance of the two techniques was similar with a slight advantage of the SOM technique. The performance of the HNIS is remarkable, in a problem that seems to be very difficult for a technique that generates non-self detectors in such a high dimensional space. It is clear that 1000 detectors are not enough to cover this space; however, the experiments showed that they were enough to train a classifier (MLP) that could effectively discriminate between normal and abnormal samples in the testing set.

5.6.3.3 Results comparison and discussion (MIT-Darpa 99)

Figure 5.13 shows the best ROC curves produced by the three techniques. The configurations are: HNIS, 5 hidden neurons; SOM, 4×6 output layer using D_∞ distance; and BNS, $r = 6$ with binary or Gray coding. The SOM method is clearly better that the other two methods. However, the other two methods also produced good results that have a detection rate over 93% with a false alarm rates as small as 1%. The HNIS method can reach a detection rate as accurate as the one produced by SOM (98%), but only if the false alarm rate is increased to 13%. Notice that this trade-off cannot be applied to the BNS. However, the BNS produces a very good detection rate (95%) with a very small false alarm rate.

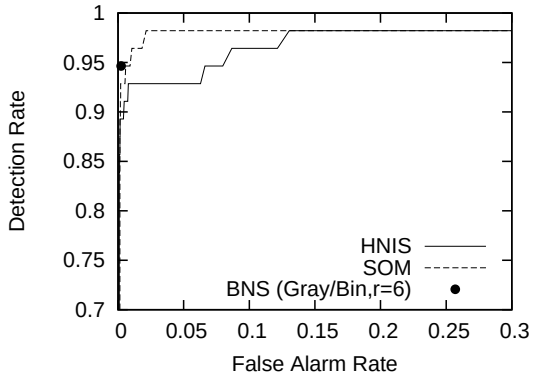


Figure 5.13: Best ROC curves produced by each method for Darpa 99 test data set.

5.6.4 Wisconsin breast cancer experiments

5.6.4.1 Data set

This data set correspond to a breast cancer data set created at the University of Wisconsin Hospitals [122]. This particular data set was obtained from the University of California Machine Learning repository[5]. Each data record is conformed by ten numerical attributes and the label (benign or malign). The data is composed by 699 records, but 16 of them have missing values. (we did not use these records.) The data was normalized to fit the interval [0,1] , and we partitioned it in two sets, training and testing. The training set contains 271 benign records. The testing set is composed of 412 mixed benign and malign records.

5.6.4.2 Experimental settings

The experimental settings for all three techniques are the same as the ones described in Section 5.6.2.1.

5.6.4.3 Results comparison and discussion

The best ROC curves produced by each method are shown in Figure 5.14. These curves are produced by the following configurations: HNIS, 18 hidden neurons and

[5]Original database at `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin`.

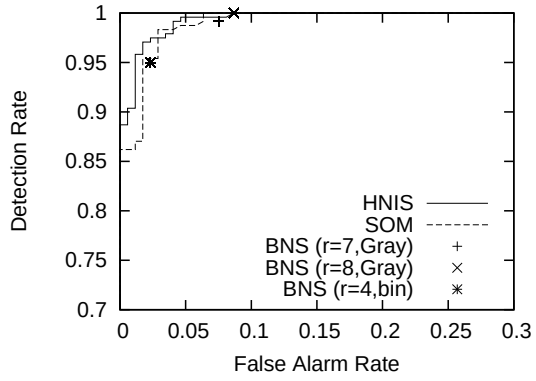


Figure 5.14: Best ROC curves produced by each method for Wisconsin breast cancer test data set.

SOM, 4×6 output layer using Euclidean distance. In the case of BNS, there are three good configurations: $r = 7$ with Gray coding, $r = 8$ with Gray coding, and $r = 4$ with binary coding. It is important to note that the points in the ROC diagram for the BNS method are generated by three different runs of the algorithm, whereas the points for the other two methods correspond to only one run in each case. All of the methods are able to produce high detection rates. The HNIS method has a slight advantage over the SOM method, mainly for small false alarm rates. For false alarm rates higher than 7%, the performance of all the methods is similar.

5.7 Summary

In this chapter, we presented a hybrid anomaly detection technique (HNIS) that combines an immune inspired algorithm, real-valued negative selection (which is also presented), and a conventional classification algorithm. This method does not use positive or negative detection. Rather, it tries to find a boundary between normal and abnormal classes.

The hybrid method is compared against binary negative selection (BNS), using the greedy algorithm with r-contiguous matching [44], and an anomaly detection technique based on self-organizing maps (SOM).

In general, the performance of HNIS and SOM methods was good. In two experiments, HNIS outperformed SOM, and in two other, the results were opposite. In all the cases, the difference was small. BNS performed well in two of the experiments; however, it failed to produce acceptable results in two other cases. The MIT-Darpa 98 data set is one of the data sets where BNS failed. This is consistent with the results reported by Kim and Bentley [92]; these results were used by them to support the claim that negative selection algorithm suffers from "severe scaling problems". However, our work shows that the problem is not with the negative selection algorithm itself, rather the kind of representation (binary) and matching rule (r-contiguous) that were used. This was also suggested by Balthrop et al. [93].

Another important characteristic of the proposed approach is that it can learn the structure of the self set using only a subset of normal samples. In some applications, mainly in change detection, it is assumed that the self set is complete; however, in many real anomaly detection applications, this is not the case. Hence, an anomaly detection algorithm must be able to produce a good approximation of the structure of the self/non-self space, even if a portion of the self set is available during training. The experiments with the Mackey-Glass data set (Section 5.6.2) are a good example of this.

Finally, the use of a more expressive representation for the detectors allows the combination of negative selection with other learning methods. Our previous work demonstrated the feasibility of combining negative selection with a classification algorithm (a MLP trained with back-propagation). A very interesting experiment would be to combine it with other immune inspired techniques, like those based on immune network theory. This would open the doors for the construction of an unified artificial system that combines different types of immune mechanisms.

Chapter 6

Mathematical Foundation of a New RNS Algorithm

The previous chapter presented a Real-Valued Negative Selection (**RNS**) algorithm that is based on some heuristics that try to distribute the antibodies (detectors) in the non-self space in order to maximize the covering. Some of the drawbacks of this approach are the following:

- The number of antibodies needed to cover the non-self space, as well as the radius of each antibody, are not known in advance; hence, it is necessary to determine them by a trial-and-error procedure.
- There is no guarantee that the algorithm will converge to an optimal or close-to-optimal space coverage with minimum overlap.

The purpose of this chapter is to present a new RNS algorithm based on mathematical foundation. This will give more criteria to setup the algorithmic parameters and to assess the expected performance.

The new algorithm is based on two main ideas:

- To estimate of the volume of the self space, which, by complementarity, is also an approximation of the volume of the non-self space. Using this volume, it is

possible to calculate how many antibodies of a given radius are needed to cover the non-self space. The algorithm used is based on Monte Carlo integration [123, 124], a method with a well-established mathematical background.

- To use a well known optimization algorithm, simulated annealing [125, 126], to find a good distribution of the antibodies that maximizes the coverage of the non-self space.

The following sections describe the details of the algorithm as well as the theoretical analysis.

6.1 Randomized real valued negative selection (RRNS) algorithm

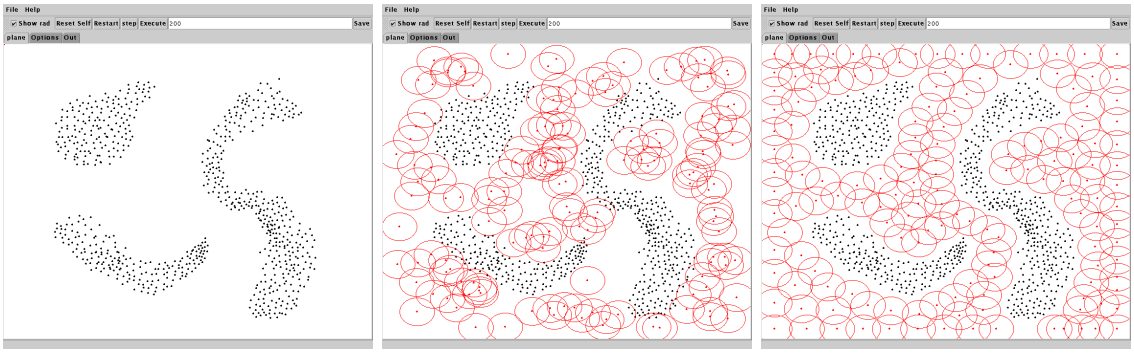
Similar to the algorithm proposed in the previous chapter (RNS, Figure 5.3 on page 101), the objective of this algorithm is to generate a set of antibodies that cover the non-self space. The approach followed to developed this algorithm is different from the approach followed in the previous chapter; the main difference is the improved mathematical foundation. The algorithm is called *randomized* because it is based on an important class of randomized algorithms called Monte Carlo methods [123, 127, 124]. Specifically, it uses *Monte Carlo integration* to estimate the volume of the self (and non-self) space and *simulated annealing* [125, 126] to optimize the distribution of antibodies in the non-self space.

The input to the algorithm is a set of samples from the self set, S'; the allowed variability in the self set, r_{self}; the antibody radius, r_{ab}; and a set of parameters, Π. The global structure of the algorithm is shown in Figure 6.1.

A typical execution of the algorithm for a two-dimensional set is shown in Figure 6.2. The algorithm is composed by two main functions: CALCULATE-INIT-ANTIBODY-SET (Section 6.2, Figure 6.4), which estimates the volume of the non-self space in order to produce a good initial set of antibodies (Figure 6.2(b)), and

```
RR-Negative-Selection(S', r_self, r_ab, Π)
   S'   :  set of self samples
 r_self :  self variability threshold
  r_ab  :  antibody radius
   Π    :  additional parameters
1: D ←Calculate-Init-Antibody-Set(S', r_self, r_ab)
2: D' ←Optimize-Antibody-Distribution(D, r_ab, S', r_self)
3: Return D'
```

Figure 6.1: Randomized real-valued negative selection (RRNS) algorithm.



(a) (b) (c)

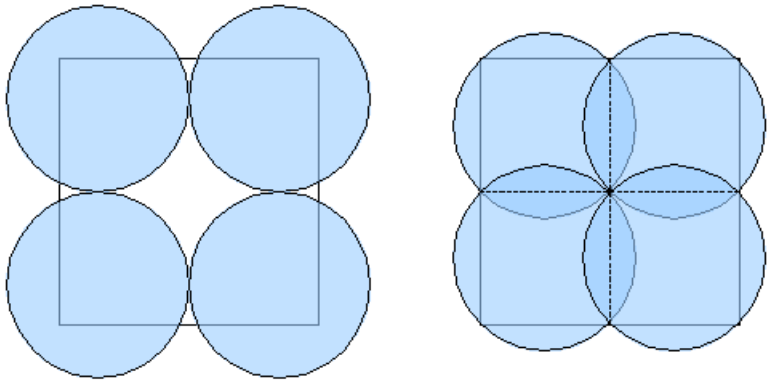
Figure 6.2: A typical execution of the RRNS algorithm (Figure 6.1) for a small 2-dimensional self set. (a) Input self set. (b) Initial set of antibodies generated by function Calculate-Init-Antibody-Set (Figure 6.4). (c) Final set of antibodies produced by function Optimize-Antibody-Distribution (Figure 6.9).

Optimize-Antibody-Distribution (Section 6.3, Figure 6.9), which distribute the antibodies uniformly in the non-self space based on simulated annealing optimization (Figure 6.2(c)). These two functions will be discussed in the following sections.

6.2 Determining the number of antibodies

Let V_d be the volume covered by an individual antibody and let $V_{\text{non-self}}$ be the volume of the non-self space. A rough approximation of the number of antibodies is given by:

$$num_{ab} = \frac{V_{\text{non-self}}}{V_d}. \tag{6.1}$$



(a) (b)

Figure 6.3: Covering of a rectangular region using circular antibodies. (a) Without overlapping. (b) With overlapping.

This is a very optimistic approximation since it does not take into account the fact that, in general, it is impossible to cover a given volume with spherical antibodies without allowing some overlapping. Figure 6.3(a) shows an example of a covering of a square using four non-overlapping antibodies. It is clear that the covering can be improved if some overlapping is allowed. Figure 6.3(b) shows such a covering with overlapping antibodies.

If overlapping is allowed, the effective covering volume is not anymore the volume of the hypersphere that defines an antibody, but a smaller value. We define the covering volume of an antibody as the volume of the inscribed hypercube. The main reason to choose this definition is that there is a straightforward way to cover an n-dimensional region using hypercubes without holes. In Figure 6.3(b), the inscribed hypercubes (in this case, squares) are drawn with dashed lines. It is easy to see that this kind of construction can be extended to a higher dimension.

Accordingly with the previous discussion, the effective volume covered by an antibody d with radius r is defined as:

$$V_d = \left(\frac{2r}{\sqrt{n}}\right)^n. \tag{6.2}$$

Using Equations 6.1 and 6.2, it is possible to calculate a good approximation of the number of antibodies with a given radius needed to cover the non-self space. This will require knowledge of the volume of the non-self space. This is the problem that we will address in the remaining part of this section.

6.2.1 Calculating the volume of the self (non-self) set

As in the previous chapters, the self/non-self space, U, corresponds to the unitary hypercube, $[0,1]^n$. Clearly, the volume of the self/non-self space is equal to 1.0; therefore, the volume of the non-self space is defined as:

$$V_{\text{non-self}} = 1 - V_{\text{self}}.$$

As discussed in Chapter 5, the input to the NS algorithm is a subset of the self set. Thus, in general, the area of the self space is not known. We will assume a model of the self set, $\widehat{S}$, that is defined in terms of a set of self samples, S'. The basic assumption in this definition is that an element that is *close enough* to a self sample is considered to be self. The closeness is specified formally by a variability threshold, r_{self}, that defines the minimum distance between a self sample and an element x, such that x can be considered part of the self set. The self set model, $\widehat{S}$, is defined as follows:

$$\widehat{S} := \left\{ x \in U \mid \exists s \in S', \|s - x\| \leq r_{\text{self}} \right\}.$$

We define V_{self} as the volume of $\widehat{S}$, which is calculated as:

$$V_{\widehat{S}} := \int_U \chi_{\widehat{S}}(x) dx \,,$$

where $\chi_{\widehat{S}}$ corresponds to the characteristic function of the set $\widehat{S}$ defined by

$$\chi_{\widehat{S}}(x) := \begin{cases} 1 & \text{if } x \in \widehat{S} \\ 0 & \text{if } x \notin \widehat{S} \end{cases}.$$

We can produce an estimate of $V_{\widehat{S}}$ using random sampling. The basic idea is to generate a sequence $\{x_i\}_{i=1..m}$ of random samples uniformly distributed in U. The expected value of $\chi_{\widehat{S}}(x_i)$ is

$$\mathrm{E}\left[\chi_{\widehat{S}}(x_i)\right] = \int_U \chi_{\widehat{S}}(x)dx = V_{\widehat{S}};$$

therefore, an estimate of $\mathrm{E}\left[\chi_{\widehat{S}}(x_i)\right]$ is automatically an estimate of $V_{\widehat{S}}$. As it is well known, a good estimate of the mean of a random variable (expected value) is the mean of a set of samples; so, we will use the average of $\left\{\chi_{\widehat{S}}(x_i)\right\}_{i=1..m}$ as an estimate, $\widehat{V_{\widehat{S}}}$, of the self volume:

$$V_{\widehat{S}} \approx \widehat{V_{\widehat{S}}} = \frac{\sum_{i=1}^{m} \chi_{\widehat{S}}(x_i)}{m}. \tag{6.3}$$

The estimation of a defined integral by averaging a set of random samples is known as *Monte Carlo integration* [123, 124]. The main advantage of this method, and contrary to other non-probabilistic methods, is that it is possible to calculate an interval of confidence for the estimated integral. We will use this approach to assess how good the volume estimate produced by Equation 6.3 is. Specifically, we want to find an error bound, ϵ, such that $Pr(|\widehat{V_{\widehat{S}}} - V_{\widehat{S}}| < \epsilon)$ is close to 1.0. In order to determine it, we can use the central limit theorem [128] that states that the distribution of the random variable

$$Z = \frac{\widehat{V_{\widehat{S}}} - V_{\widehat{S}}}{\sqrt{var(\widehat{V_{\widehat{S}}})}} = \frac{\widehat{V_{\widehat{S}}} - V_{\widehat{S}}}{\sqrt{var(\chi_{\widehat{S}}(x_i))/m}},$$

where $var(\cdot)$ is the variance, tends to $\mathcal{N}(0,1)$ (standard normal distribution) when $m \to \infty$. Therefore,

$$\begin{aligned} & Pr(-3 \leq Z \leq 3) \approx 0.998 \\ \Rightarrow \quad & Pr(-3 \leq \tfrac{\widehat{V_{\widehat{S}}} - V_{\widehat{S}}}{\sqrt{var(\chi_{\widehat{S}}(x_i))/m}} \leq 3) \approx 0.998 \\ \Rightarrow \quad & Pr(|\widehat{V_{\widehat{S}}} - V_{\widehat{S}}| \leq 3\sqrt{var(\chi_{\widehat{S}}(x_i))/m}) \approx 0.998. \end{aligned} \tag{6.4}$$

The variance of $\chi_{\widehat{S}}(x_i)$ is

$$var(\chi_{\widehat{S}}(x_i)) = \mathrm{E}\left[\left(\chi_{\widehat{S}}(x_i)\right)^2\right] - \mathrm{E}\left[\chi_{\widehat{S}}(x_i)\right]^2 = V_{\widehat{S}} - V_{\widehat{S}}^2. \tag{6.5}$$

Using Equations 6.4 and 6.5, we can calculate a superior bound, ϵ, for the error:

$$\epsilon = 3\sqrt{\frac{var(\chi_{\widehat{S}}(x_i))}{m}} = 3\sqrt{\frac{V_{\widehat{S}} - V_{\widehat{S}}^2}{m}} \approx 3\sqrt{\frac{\widehat{V_{\widehat{S}}} - \widehat{V_{\widehat{S}}}^2}{m}}. \tag{6.6}$$

Finally, the confidence interval for the self volume estimate, $\widehat{V_{\widehat{S}}}$, is given by:

$$Pr\left(|\widehat{V_{\widehat{S}}} - V_{\widehat{S}}| < 3\sqrt{\frac{\widehat{V_{\widehat{S}}} - \widehat{V_{\widehat{S}}}^2}{m}}\right) \approx 0.998. \tag{6.7}$$

6.2.2 Algorithm to calculate an initial set of antibodies

Now that we know how to calculate the area of the self (non-self) space, it is straightforward to calculate the number of antibodies that are needed to cover the non-self space and to generate an initial set of antibodies located in the non-self space. The process is shown in Figure 6.4.

The algorithm receives as input the set of samples from self (S'), the variability radius of the self set (r_{self}), the radius of each antibody (r_{ab}), the maximum allowed error (ϵ_{max}), and a minimum number of iterations that have to be performed (m_{min}). The purpose of the last parameter, m_{min}, is to produce a good initial estimate of the error (ϵ) by enforcing a minimum number of iterations. This prevents a premature stop of the algorithm due to a poor initial estimation of ϵ. Notice that the algorithm can be easily modified to receive as input the number of antibodies instead of the antibody radius (r_{ab}). In that case, line 13 must be replaced by

$$r_{ab} \leftarrow \sqrt[n]{\frac{1 - \widehat{V_{\widehat{S}}}}{num_{ab}}} \cdot \frac{\sqrt{n}}{2}. \tag{6.8}$$

```
CALCULATE-INIT-ANTIBODY-SET(S', r_self, r_ab, ε_max, init_iter)
    S'   :  set of self samples
  r_self :  self variability threshold
   r_ab  :  antibody radius
  ε_max  :  maximum allowed error
  m_min  :  initial number of iterations
    n    :  dimension of the self/non-self space
 1: num_hits ← 0
 2: m ← 0
 3: Repeat
 4:    m ← m + 1
 5:    x ←uniformly distributed random sample from [1,0]^n
 6:    y ←NEAREST-NEIGHBOR(S', x)
 7:    If ||x − y|| ≤ r_self
 8:       Then num_hits ← num_hits + 1
 9:    EndIf
10:    V̂_Ŝ ← num_hits / m                          ▷ Eq. 6.3
11:    ε ← 3 sqrt( (V̂_Ŝ − V̂_Ŝ^2) / m )            ▷ Eq. 6.6
12: Until m ≥ m_min and ε ≤ ε_max
13: num_ab ← ⌊ (1 − V̂_Ŝ) / ((2r_ab/√n)^n) ⌋        ▷ Eq. 6.2
14: D ← ∅
15: Repeat
16:    x ←uniformly distributed random sample from [1,0]^n
17:    y ←NEAREST-NEIGHBOR(S', x)
18:    If ||x − y|| ≥ r_self
19:       Then D ← D ∪ {x}
20:    EndIf
21: Until |D| = num_ab
22: Return D
```

Figure 6.4: Algorithm to calculate the initial antibody set.

The function NEAREST-NEIGHBOR used in lines 6 and 17 is defined as follows:

$$\text{Nearest-neighbor}(S', x) = \arg\min_{y \in S'} \|x - y\| .$$

If we perform a sequential scan on S', the function can be calculated in $O(N)$ time, where $N = |S'|$. It is possible to speed-up this query by using a spatial access method such as an R*-tree [129, 130]. Despite the fact that the worst case time complexity of nearest neighbor queries using this kind of methods is still $O(N)$, they are much more efficient than sequential scanning for large values of N.

The number of iterations of the loop in lines 3 to 12, m, is determined by the area of the self set and the maximum error as follows:

$$m = \frac{9(V_{\widehat{S}} - V_{\widehat{S}}^2)}{\epsilon^2}.$$

Notice that the quantity $V_{\widehat{S}} - V_{\widehat{S}}^2$ is maximum when $V_{\widehat{S}} = 0.5$; hence the value $9/(4\epsilon^2)$ is a strict upper bound for m.

The number of iterations of the loop in lines 15 to 21 depends on the number of antibodies to be generated, num_{ab}, and the probability that a randomly generated antibody will be in the non-self space, which is same as the non-self volume, $1 - V_{\widehat{S}}$. The expected number of iterations is thus $O(\frac{num_{ab}}{1-V_{\widehat{S}}})$. In general, the number of iterations for this loop is not expected to be large since, for practical problems, the number of antibodies is not going to be large, and the volume of the non-self space $(1 - V_{\widehat{S}})$ is not expected to be close to 0. Furthermore, it is possible to store some of the points that fail the test in line 7 in D; this will reduce the number of iterations needed to generate num_{ab} antibodies. In conclusion, the time of the algorithm is expected to be dominated by the first part (lines 1 to 12), i.e. the expected time is in the order $O(\frac{N}{\epsilon^2})$, where $N = |S'|$.

6.3 Improving the antibody distribution

Figure 6.2(b) shows a typical distribution of the antibodies generated by the algorithm from the previous section (Figure 6.4) for a small two-dimensional self set. Clearly, the distribution is far from optimal, which is not surprising since the unique goal of the algorithm is just to produce a set of antibodies that do not match any self point. The purpose of this section is to develop a procedure that improves the distribution of the antibodies produced by the CALCULATE-INIT-ANTIBODY-SET algorithm (Figure 6.4) in order to optimize the covering of the non-self space.

The problem of finding a good distribution of the antibodies can be better stated as an optimization problem:

Maximize:

$$V(D) = Volume\left\{x \in U \mid \exists d \in D, \|x - d\| \leq r_{ab}\right\}, \tag{6.9}$$

restricted to:

$$\{s \in S' \mid \exists d \in D, \|s - d\| \leq r_{ab}\} = \emptyset \text{ (not covering of self)}, \tag{6.10}$$

where,

D : set of antibodies with a fix cardinality num_{ab},

r_{ab} : antibody radius, and

S' : input self set.

The function defined in Equation 6.9 represents the amount of the self/non-self space covered by a set of antibodies, D, which corresponds to the volume covered by the union of the hyper-spheres associated with each antibody. The restriction specified in Equation 6.10 tells that no antibody can match any self point.

The evaluation of the function $V(D)$ can be a costly process; in fact, the only practical way to do it is using a Monte Carlo integration method similar to the one used in the previous section (6.2). Instead, we will use a simplified version of this optimization problem, which we will show, experimentally, to be equivalent.

The remaining part of this section describes an optimization algorithm to solve this problem. The technique is based on a very well known Monte Carlo based optimization method, *simulated annealing*, which is adapted to solve this particular problem.

This is not the first time that simulated annealing has been used in an AIS. De Castro and Von Zuben [53] proposed a technique to initialize feed-forward neural networks weights. The basic idea is to represent the network weights by antibodies which correspond to n-dimensional real-valued vectors. The antibodies are dispersed in the space by maximizing an energy function that takes into account the inverse of the inter-antibody affinity. This approach is substantially different to the one proposed here; it does not use the concept of self/non-self distinction, and its main goal is producing diversity instead of performing anomaly detection.

6.3.1 Simulated annealing

The simulated annealing technique was initially proposed by Kirkpatrick et al. [125] borrowing inspiration from the physical annealing of solids. The physical process is more or less as follows: a solid is heated to a high temperature, then, it is slowly cooled until some desired properties of the solid are obtained; these properties are related to a low energy state.

In the algorithm, the energy corresponds to the function to minimize, $C(s)$, whose domain is the space of states of a system. The system is randomly perturbed by moving it from the current state, s_i, to a new state, s_j. If $C(s_j) < C(s_i)$, the transition is accepted; otherwise, its acceptance is defined by a random process. The probability of accepting this transition is a function of the temperature: the higher the temperature, the higher the probability of accepting a worse state. This step is repeated a number of times until the system reaches *thermal equilibrium*. This perturbation process is known as the Metropolis algorithm [124, 131], and it belongs

to a broader class of algorithms called Monte Carlo methods [124]. The simulated annealing algorithm is shown in Figure 6.5.

In order to apply this algorithm to solve a specific problem, it is necessary to define the following elements:

- The set of possible configurations (states) of the system, $States$.
- The neighborhood of each state. For each state $s_i \in States$, it is defined a set $N_{s_i} \subseteq States$ that contains all the states where it is possible to move from s_i. It is indispensable that $\forall s_i, s_j \in States\ s_j \in N_{s_i} \Leftrightarrow s_i \in N_{s_j}$, i.e. if it is possible to perform a transition from a state i to a state j, it has to be possible to move back from j to i.
- A cost function $C : States \rightarrow \mathbb{R}$.
- A stopping criterion for the inner loop (*thermal equilibrium*). The inner loop can be seen as a sequence of transitions on a Markov chain [132]. The equilibrium is reached when the probability distribution of the states approaches the Boltzmann distribution. In a practical application, a common heuristic is to perform a number of iterations that depends on the size of the problem [126].
- A cooling schedule, i.e. a process that determines the sequence of temperatures: $T_0, T_1, ..., T_m$. This can be decomposed in:
  - An initial temperature T_0. The idea is to have an enough high temperature such that the acceptance ratio, χ_0, is close to 1 (in [125], $\chi_0 = 0.8$). Johnson et al. [133] proposed the following rule:

$$T_0 = \frac{\overline{\Delta C}^{(+)}}{\ln(\chi_0^{-1})}, \tag{6.11}$$

    where $\overline{\Delta C}^{(+)}$ is the average increase in cost for a number of random transitions.

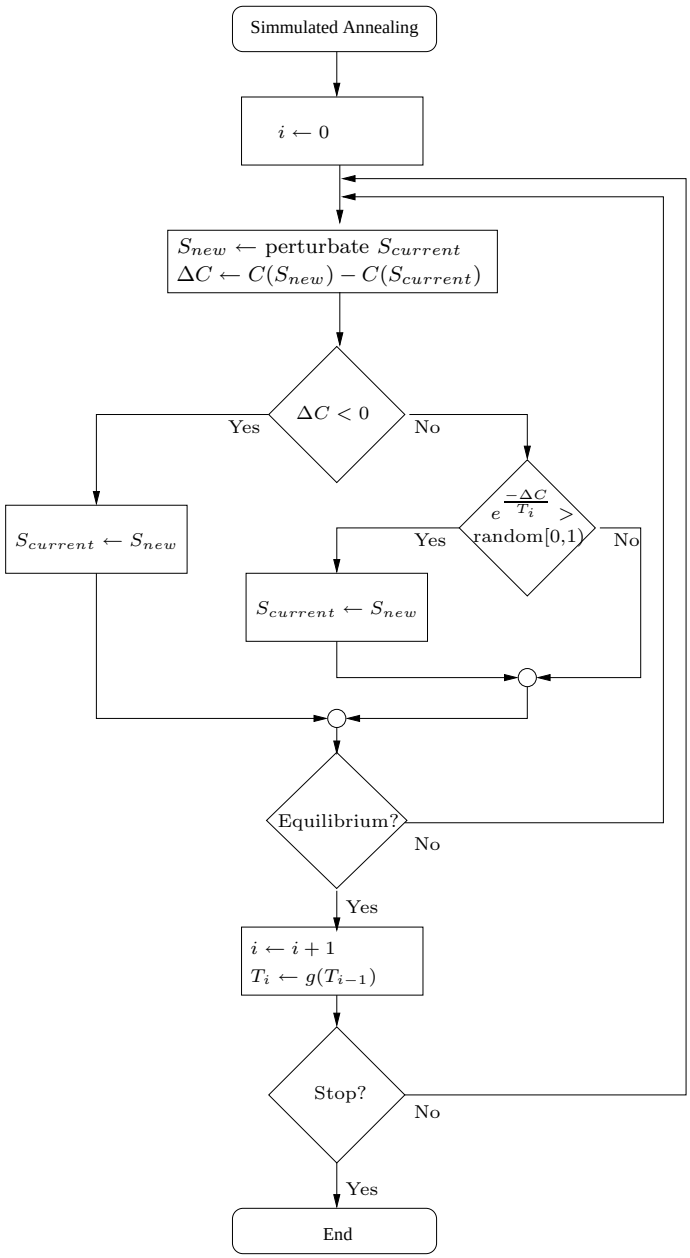


Figure 6.5: Simulated annealing algorithm.

– Decrement of the temperature, $T_{i+1} \leftarrow g(T_i)$. A frequently used update rule is given by

$$T_{i+1} \leftarrow \alpha \cdot T_i, \tag{6.12}$$

where $\alpha \in [0.8, 0.99]$.

– A stopping criterion for the outer loop. A simple option is to specify a fixed number of iterations. A more elaborated strategy will look for the successive changes in the configuration and stop if a given number of consecutive configuration is the same.

6.3.2 An algorithm to optimize the volume covered by the antibody set

As we mentioned at the beginning of this section, the algorithm that optimizes the covering of the non-self space is based on the simulated annealing technique. In order to describe it, we will follow the list of elements presented in the previous subsection.

6.3.2.1 Set of system configurations

The main input to this algorithm is the set of antibodies produced by the CALCULATE-INIT-ANTIBODY-SET algorithm (Figure 6.4). The algorithm modifies the coordinates of the antibodies looking for a configuration that optimizes the covering of non-self. In consequence, the configuration of the system is given by the coordinates of the antibody set. Notice that the number of antibodies is fixed, no antibodies are created or eliminated in this algorithm. The set of system states is defined as follows:

$$States = \{(d_1, \ldots d_{num_{antib}}) \,|\, d_i \in [0,1]^n\}.$$

6.3.2.2 State neighborhood

A perturbation of the system corresponds to a change on the position of an individual antibody to a place close to its current position. The neighborhood of a state $s = (d_1, \ldots d_{num_{ab}})$ is defined as follows:

$$N_s = \{(\overline{d}_1, \ldots \overline{d}_{num_{ab}}) \,|\, \exists i \, \forall j \neq i, \, \overline{d}_j = d_j \text{ and } \left\|\overline{d}_i - d_i\right\| \leq r_{pert}\}, \tag{6.13}$$

where r_{pert} is a parameter of the algorithm that represents the maximum distance to move an individual antibody. In other words, a state s' is a neighbor of s ($s' \in N_s$), if s' differs from s only in one antibody, d_i, which has been moved maximum an r_{pert} distance.

6.3.2.3 The cost function

The original function to optimize corresponds to the volume covered by the antibody set (Equation 6.9); however, to calculate it can be very costly. Therefore, we need another function which is easier to calculate, and such that its optimization corresponds to the optimization of the covered volume. Intuitively, to maximize the covering produced by a set of antibodies, it is necessary to reduce their overlapping, i.e., to increase the inter-antibody distance. The following equation defines an approximate measure of overlapping between two antibodies:

$$\text{Overlapping}(d_i, d_j) = e^{\frac{-\left\|d_i - d_j\right\|^2}{r_{ab}^2}}. \tag{6.14}$$

The overlapping function is shown in Figure 6.6 along with two antibodies with radius $r_{ab} = 1$. The maximum value, 1, is reached when the distance between the two antibodies is 0. When the distance is equal to $2r_{ab}$, the value of the function is very close to 0. Notice that this function can be interpreted as the matching function of the antibody.

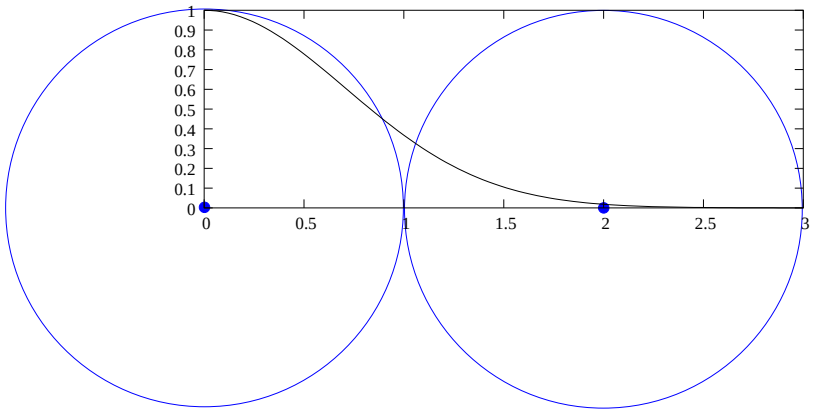


Figure 6.6: Overlapping function for two antibodies with radius $r_{ab} = 1$.

Based on Equation 6.14, the overlapping of a set $D = \{d_1, \ldots, d_{num_{ab}}\}$ of antibodies is defined as

$$\text{Overlapping}(D) = \sum_{i \neq j} e^{\frac{-\left\|d_i - d_j\right\|^2}{r_{ab}^2}}, \; i, j = 1, \ldots num_{ab}. \tag{6.15}$$

Now, the question is if minimizing $Overlapping(D)$ is the same as maximizing $V(D)$ (Equation 6.9). In general, it is not true; however, we will show in the next section that in the practice they are equivalent.

The original optimization problem includes a restriction, not covering of the self set (Equation 6.10). Simulated annealing does not provide a direct way to include restrictions; therefore, it is necessary to include a term in the cost function that penalizes configurations that violate this restriction. Then, the function to optimize is defined as follows:

$$C(D) = \text{Overlapping}(D) + \beta \cdot \text{SelfCovering}(D), \tag{6.16}$$

where, the second term corresponds to the penalization factor for violating the no-self-covering restriction, and is defined by

$$\text{SelfCovering}(D) = \sum_{s \in S'} \sum_{d \in D} e^{\frac{-\|d-s\|^2}{\left(\frac{r_{ab}+r_{self}}{2}\right)^2}}. \tag{6.17}$$

CALCULATE-COST-DIFFERENCE(D, $index$, d, r_{ab}, S', r_{self}, β)

$D = \{d_1, \ldots, d_{num_{ab}}\}$: initial antibody set
$index$: index of the antibody affected by the transition
d: new position of the antibody
r_{ab}: antibody radius
S': set of self samples
r_{self}: self variability threshold
β: Self covering penalization coefficient

1: $SelfCovering \leftarrow 0$
2: For each $s \in S'$
3: $\quad SelfCovering \leftarrow e^{\frac{-\|d-s\|^2}{\left(\frac{r_{antib}+r_{self}}{2}\right)^2}} - e^{\frac{-\|d_{index}-s\|^2}{\left(\frac{r_{antib}+r_{self}}{2}\right)^2}}$
4: EndFor
5: $Overlapping \leftarrow 0$
6: For each $d_i \in (D - \{d_{index}\})$
7: $\quad Overlapping \leftarrow e^{\frac{-\|d-d_i\|^2}{r_{ab}^2}} - e^{\frac{-\|d_{index}-d_i\|^2}{r_{ab}^2}}$
8: EndFor
9: Return $Overlapping + \beta \cdot SelfCovering$

Figure 6.7: Algorithm to calculate the cost difference produced by a transition that changes the position of an antibody d_{index} to d.

Notice that this function is based on the same principle used to define the *Overlapping* function (Equation 6.15). Each individual term on the sum measures the amount of matching between an antibody and a self element.

The term β in equation 6.16 specifies the relative importance of self-covering with respect to the inter-antibody overlapping. It controls the amount of penalization in the cost function caused by violating the no-self-covering restriction.

An advantage of this cost function is that in each step of the algorithm it is not necessary to calculate all the terms in Equations 6.15 and 6.16. It is only required to evaluate the terms that involve the antibodies affected by the transition. The function that calculates the difference of the cost function for a given transition is presented in Figure 6.7. The main inputs are the set of antibodies, D; the index of the antibody that is going to be moved, $index$; and the new position of this antibody, d. The complexity of the algorithm is clearly $O(|S'| + num_{antib})$.

6.3.2.4 Stopping criterion for the inner loop

A common heuristic [126] to determine the number of iterations for the inner loop is to make it proportional to the size of the problem to solve. Another heuristic is to impose a minimum limit to the number of accepted transitions, *accepted_transitions* [126]. In our implementation, we chose to use a combination of the two heuristics: we imposed a minimum value of accepted transitions which depends proportionally on the size of the problem (given by the number of antibodies, num_{ab}). Additionally, we imposed a maximum number of iterations for the inner loop, $inner_{max}$ to avoid extremely long Markov chains [133, 125, 126]. The two values are defined as follows:

$$accepted\_transitions = \eta_{min} \cdot num_{ab}, \tag{6.18}$$

$$inner_{max} = 2 \cdot \eta_{min} \cdot num_{ab}, \tag{6.19}$$

where, η_{min} is a input parameter to the algorithm. This parameter can be interpreted as the expected number of times that each antibody is going to be moved for a given value of the temperature.

6.3.2.5 Cooling schedule

The initial value of the temperature is calculated using Equation 6.11 as suggested by Johnson et al. [133, 126]. In order to calculate $\overline{\Delta C}^{(+)}$, we generate a fix number of random transitions on the initial antibody configuration and calculate the average of those that produce an increase in the energy. This process is shown in Figure 6.8.

NUM_ITER is a constant that determines the number of iterations. It will determine the precision of the estimate. Since we do not need a high precision for the initial temperature, a value of $NUM\_ITER = 100$ will be enough. The constant χ_0 refers to the desired initial acceptance rate; the idea is to make it close to 1. We chose to use $\chi_0 = 0.8$ as it was used by Kirkpatrick et al. [125]. The complexity of

```
CALCULATE-INIT-T() D, r_antib, S', r_self, r_pert, β)
 D = {d_1, ..., d_num_antib}:  initial antibody set
                    r_ab:  antibody radius
                      S':  set of self samples
                  r_self:  self variability threshold
                  r_pert:  neighborhood radius
                       β:  Self covering penalization
                           coefficient
Constants
 NUM_ITER:  number of iterations
        χ_0:  acceptance rate
 1: ΔC^(+) ← 0
 2: For i ← 1 to NUM_ITER
 3:    index ← random element {1, .., num_ab}
 4:    d ← random element {v ∈ [0,1]^n | ||d_index − v|| ≤ r_pert}
 5:    ΔC ← CALCULATE-COST-DIFFERENCE(D, index, d, r_ab, S', r_self, β)
 6:    If  ΔC > 0 Then
 7:       ΔC^(+) ← ΔC^(+) + ΔC
 8:    EndIf
 9: EndFor
10: ΔC̄^(+) ← ΔC̄^(+) / NUM_ITER
11: Return ΔC̄^(+) / ln(χ_0^{-1})
```

Figure 6.8: Algorithm to calculate the initial value of the temperature, T_0.

the CALCULATE-INIT-T function (Figure 6.8) is then the same as the complexity of the function CALCULATE-COST-DIFFERENCE (Figure 6.7), that is, $O(|S'| + num_{ab})$.

The temperature is decreased in each iteration of the outer loop using Equation 6.12. The neighbor radius, r_{pert}, is also decreased along with the temperature using an analogous updating rule:

$$r_{pert_i} \leftarrow \alpha_{pert} \cdot r_{pert_{i-1}}. \tag{6.20}$$

The number of outer loop iterations is given as a parameter to the algorithm.

6.3.2.6 Optimization algorithm for antibody distribution

The antibody distribution algorithm is shown in Figure 6.9. The main inputs to the algorithm are the initial antibody set (generated by the CALCULATE-INIT-ANTIBODY-SET algorithm, Figure 6.4), D; the set of self samples, S'; and the number of iterations, num_{iter}. The shape of antibodies and self elements is determined by the antibody radius, r_{ab}, and the self variability threshold, r_{self}, respectively. The number of iterations on the inner loop (lines 5 to 20) is controlled by the parameter, η_{min}, which expresses the minimum number of accepted transitions as a percentage of the number of antibodies (see Equation 6.18). The temperature decay rate, α, and the neighborhood radius decay rate, α_{pert}, control how the temperature and the neighborhood radius are going to be changed (see Equations 6.12 and 6.20) in each iteration of the outer loop. Finally, the parameter β specifies the relative importance of covering self points when calculating the cost function (see Figure 6.7).

The number of iterations in the inner loop (lines 5 to 20) is at most $2 \cdot \eta_{min} \cdot num_{ab}$. The time of each iteration is dominated by the time of the CALCULATE-COST-DIFFERENCE function, which is $O(|S'| + num_{ab})$ (see Subsection 6.3.2.3 on page 135) . Therefore, the time of each iteration of the outer loop (lines 3 to 23) has order $O(num_{ab} \cdot (|S'| + num_{ab}))$. In general, we expect the number of antibodies to

```
OPTIMIZE-ANTIBODY-
DISTRIBUTION(D, r_ab, S', r_self, num_iter, η_coef, , α, α_pert, β)
 D = {d_1, ..., d_num_antib}  :  initial antibody set
                        S'    :  set of self samples
                  num_iter    :  number of iterations
                     r_antib  :  antibody radius
                     r_self   :  self variability threshold
                     η_min    :  minimum accepted transitions %
                        α     :  Temperature decay rate
                     α_pert   :  Neighborhood radius decay rate
                        β     :  Self covering importance coefficient
 1: r_pert ← 2 · r_antib
 2: T ← CALCULATE-INIT-T(D, r_antib, S', r_self, r_pert, β)
 3: For i ← 1 to num_iter
 4:    η ← 0, steps ← 0
 5:    Repeat
 6:       index ← random element {1, .., num_antib}
 7:       d ← random element {v ∈ [0,1]^n | ||d_index − v|| ≤ r_pert}
 8:       ΔC ← CALCULATE-COST-DIFFERENCE(D, index, d, r_ab, S', r_self, β)
 9:       If ΔC < 0
10:         Then   ▷ accept transition
11:           η ← η + 1
12:           d_index ← d
13:         Else
14:           If e^(−ΔC/T) > random [0,1)
15:             Then   ▷ accept transition
16:               η ← η + 1
17:               d_index ← d
18:           EndIf
19:       EndIf
20:    Until η ≥ η_min · num_ab or steps > 2 · η_min · num_ab
21:    T ← α · T
22:    r_pert ← α_pert · r_pert
23: EndFor
24: Return D
```

Figure 6.9: Algorithm to optimize the distribution of antibodies in order to improve the covering of the non-self space.

be, at most, in the same order of magnitude as the size of the self set. Thus, the time of the algorithm is expected to be $O(num_{iter} \cdot num_{ab} \cdot |S'|)$.

6.3.2.7 Algorithm convergence

The simulated annealing algorithm can be seem as the simulation of an inhomogeneous Markov chain [126], that is, a Markov chain where the transition probability matrix changes with the time. In this case, the transition probability matrix change is due to the decrease of the temperature.

Several researchers have proved that the simulated annealing algorithm converges to an optimal state if the following conditions are satisfied:

1. the Markov chain associated with the next state generation probability matrix, G_{ij}, is irreducible and
2. the cooling schedule is such that $T_i = \frac{\Gamma}{\ln(i+1)}$, where Γ is a problem dependent constant.

The matrix G_{ij} represents the probability of generating the next state s_j from the current state s_i and is defined by the neighborhood sets in the following way:

$$G_{ij} = \begin{cases} \frac{1}{|N_{s_i}|}, & \text{if } s_j \in N_{s_i} \\ 0 & \text{otherwise} \end{cases}.$$

The irreducibility property indicates that for all pairs of states (s_i, s_j) there is a positive probability of reaching s_j from s_i in a finite number of transitions:

$$\forall s_i, s_j, \exists m \, : \, (G^m)_{ij} > 0,$$

which is equivalent to

$$\forall s_i, s_j, \exists l_1, \ldots, l_m \, : \, s_{l_1} \in N_{s_i} \wedge s_j \in N_{s_{l_m}} \wedge (\forall k, 1 \leq k < m \, : \, s_{l_{k+1}} \in N_{s_{l_k}}).$$

It is not difficult to verify that this is satisfied by the neighborhood sets defined in Equation 6.13.

The second condition is clearly not satisfied by the algorithm. However, this is easy to solve by changing the temperature updating rule. This updating rule is very slow, and, in general, of no practical use for real applications. This means that there is a trade-off between optimality and efficiency, which we preferred to solve in favor of the efficiency.

6.3.2.8 Implementation considerations

The most costly operation in OPTIMIZE-ANTIBODY-DISTRIBUTION function (Figure 6.9) is to calculate ΔC (a task that is performed by CALCULATE-COST-DIFFERENCE function, Figure 6.7). The reason is that it has to visit each element in both the self set and the antibody set in order to calculate the change in the measures $SelfCovering(D)$ and $Overalpping(D)$. It is important to notice that, in general, each individual term of the sum (in Equations 6.14 and 6.17) has a significant value if it corresponds to the overlapping of two antibodies (or one antibody and a self element) that are close enough; otherwise, the term is almost 0. Specifically, if $\|d^1 - d^2\| > 2 \cdot r_{ab}$, Overlapping$(d^1, d^2) \approx 0$ (see Equation 6.14). This means that to calculate the change in $Overlapping(D)$, when a specific antibody d^i is moved to a new position d, it is enough to take into account those antibodies d^j such that

$$\min(\left\|d^i - d^j\right\|, \left\|d - d^j\right\|) < 2 \cdot r_{ab}. \tag{6.21}$$

In order to take advantage of this, it is necessary to use an appropriate data structure that performs range queries efficiently. A multidimensional access method [134] such as an R*-tree [129] will do the work. To execute the query more efficiently, it is necessary to transform it in a rectangular query. In this case, we take the minimum hyper-rectangle that encloses the region defined by Equation 6.21. This is illustrated in Figure 6.10. The hyper-rectangle is defined by the coordinates of

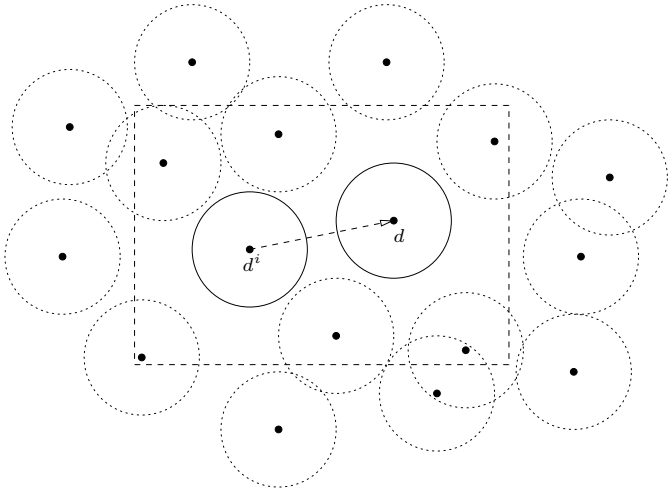


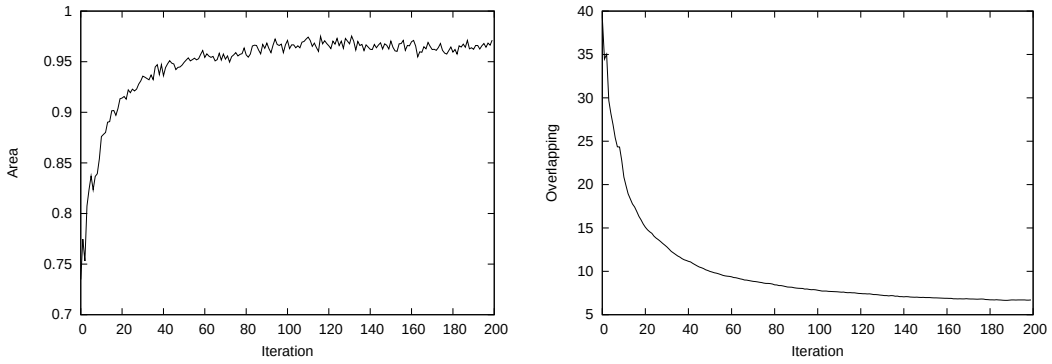
Figure 6.10: The dashed rectangle enclose the antibodies affected by the movement of antibody d^i to a new position d. The rectangle corresponds to the boundaries of the region defined by Equation 6.21.

the two opposite corners: $(\min(d_1^i, d_1) - 2 \cdot r_{antib}, \ldots, \min(d_n^i, d_n) - 2 \cdot r_{antib})$ and $(\max(d_1^i, d_1) + 2 \cdot r_{antib}, \ldots, \max(d_n^i, d_n) + 2 \cdot r_{antib})$, where the subindex refer to the individual components on each dimension.

Notice that the use of these types of data structures will speed up the algorithm, only if the size of the self and the antibody sets is large enough to compensate the overhead of creating and maintaining the structure. Also, this does not change the complexity of the algorithm since the worst case time of a range query is still linear on the size of the set.

6.4 RRNS experimentation

The purpose of this section is to validate experimentally some of the assumptions made while developing the algorithm presented in this chapter. Section 6.3 formulates the problem of antibody distribution as an optimization problem corresponding to maximizing the non-self volume covered by a set of antibodies ($V(D)$, Equations 6.9 and 6.10). The Optimize-Antibody-Distribution algorithm (Figure 6.9) solves a modified optimization problem: to minimize the function $C(D)$ defined by Equation 6.16. This function is composed by a term that measures the amount of overlapping



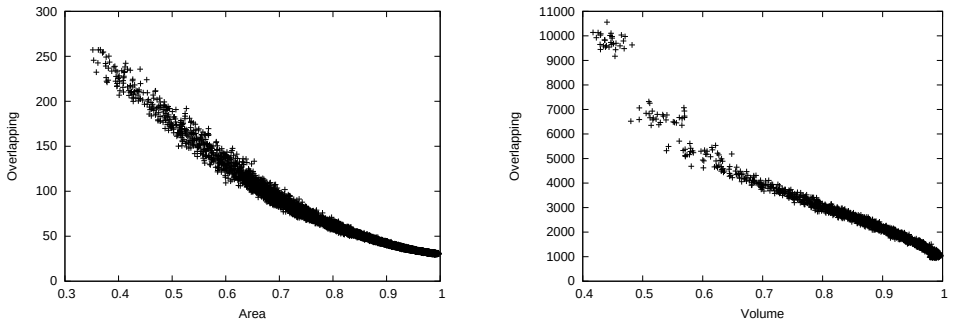
(a) (b)

Figure 6.11: (a) Evolution of the area covered by the antibodies. (b) Evolution of the inter-antibody overlapping measured by Equation 6.15.

between antibodies and a term that penalizes the covering of self points. The main assumption is that minimizing $C(D)$ is approximately equivalent to maximizing $V(D)$. The intuition behind this assumption is that the less overlapping in a set of antibodies, the larger the volume covered by them.

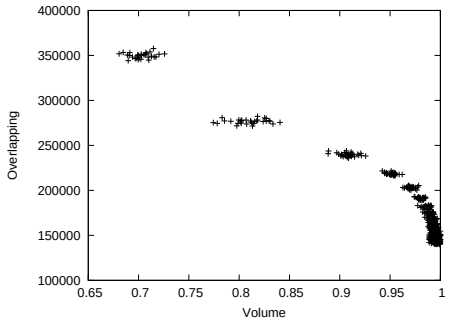
Figure 6.11 shows the evolution of the area covered by a set of antibodies and their overlapping when the Optimize-Antibody-Distribution (Figure 6.9) is applied to an initial set of random antibodies in the unitary square. The overlapping, which is the objective function minimized by the algorithm, goes down with the successive iterations. This means that the antibodies are moving apart. This causes an increase in the area covered by them, as shown in Figure 6.11(a). The area curve is not as smooth as the overlapping curve; this can be explained by the fact that the area is estimated (using Monte Carlo integration, $\epsilon = 0.01$), whereas the overlapping is calculated exactly.

The previous experiment suggests that, in fact, the algorithm is able to maximize the area covered by minimizing the inter-antibody overlapping. However, this is just one experiment in a 2-dimensional space. In order to build a stronger experimental evidence, we performed the following experiment: a random set of antibodies is generated close to the center of the unitary hypercube, then the function Optimize-



(a)

(b)



(c)

Figure 6.12: The graphics show the overlapping-versus-volume relation for a set of antibodies produced by the successive application of Optimize-Antibody-Distribution function (Figure 6.9). (a) Dimension = 2. (b) Dimension = 5. (c) Dimension = 10.

Antibody-Distribution (Figure 6.9) is applied for a given number of iterations, the volume covered and the inter-antibody overlapping (Equation 6.15) are measured; this process is repeated 30 times, each time starting with a new random set of antibodies.

Figure 6.12 shows the overlapping-versus-volume graphics corresponding to the data generated by the experiment for space dimension 2, 5, and 10. It is easy to see that there is a clear inverse relationship between the volume covered by a set of antibodies and their inter-antibody overlapping. As it is shown by the graphics, the relationship is not necessarily linear; however, it does not affect the algorithm since it is enough that the volume increases monotonically when the overlapping decreases.

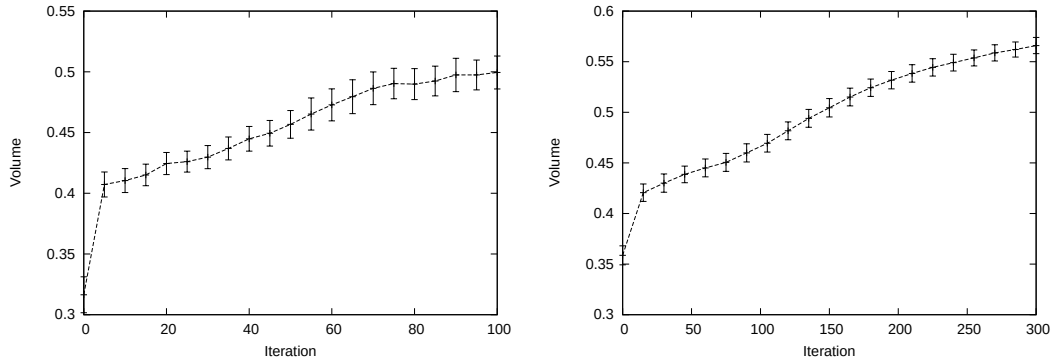
Table 6.1: Parameter values for the RRNS and RNS algorithms

RRNS parameters	
num_{antib}	100
r_{self}	0.1
ϵ	0.005
η_{min}	0.3
α	0.95
α_{pert}	0.95
β	1
RNS parameters	
r	0.04838
η	1
t	5
k	1

An interesting question is: how does the new algorithm (RRNS) compare to the previous algorithm (RNS) in terms of the optimization of the volume covered by the set of antibodies? It is important to take into account that the RNS algorithm was not developed to optimize explicitly a function, neither the volume nor the overlapping. The algorithm is based on heuristic rules that try to move the antibodies away from each other and from the self points. An indirect result of this is an increase in the non-self area covered by the set of antibodies. Therefore, we expect the RRNS algorithm to perform better than the RNS algorithm in terms of the optimization of the area covered by the generated set of antibodies.

To perform the comparison, we chose a simple data set that allows us to perform many runs of both algorithms. The multiple runs eliminate any dependence of the results on the initial conditions. The data set to use corresponds to a subset of the Mackey-Glass time series data set (Subsection 4.6.1.1), which includes the first and the fourth features (the data set is shown in Figure 3.3(b)). This set is used as input to both algorithms along with the parameters specified in Table 6.1.

Notice that the RRNS is able to calculate the antibody radius if the number of antibodies is given (Equation 6.8); this is not the case for RNS. Therefore, to make the comparison fairer, we used the antibody radius calculated by the RRNS algorithm



(a) (b)

Figure 6.13: Evolution of the non-self covered volume when RNS and RRNS algorithms are applied to the same self set. The points represent the average of 30 experiments and the length of the vertical lines represent three times the standard deviation. (a) Real-valued negative selection (RNS). (b) Randomized real-valued negative selection (RRNS).

as input for the RNS algorithm. The parameters shown are the ones that produced the best results for each algorithm.

The algorithms were run for a fix number of iterations (300 for RRNS and 100 for RNS). After each iteration, the volume covered by the set of antibodies was calculated using a Monte Carlo integration method similar to the one used in the Calculate-Init-Antibody-Set algorithm (Figure 6.4). In this case, the value of the error was ϵ=0.005. The process was repeated 30 times. Figure 6.13 shows the evolution of the covered volume for each one of the algorithms.

The points in the curve represent the average volume for the 30 experiments, and the length of vertical lines correspond to three times the standard deviation. In both cases, the covered volume increases with the successive iterations. The RRNS algorithm produces a larger covering volume, as was expected.

According to Figure 6.13, the RRNS uses more iterations. However, the two algorithms are very different, and the type and the amount of calculations performed during each iteration varies from one algorithm to the other. A better comparison will use the time instead of the iteration number. Figure 6.14 shows such a comparison.

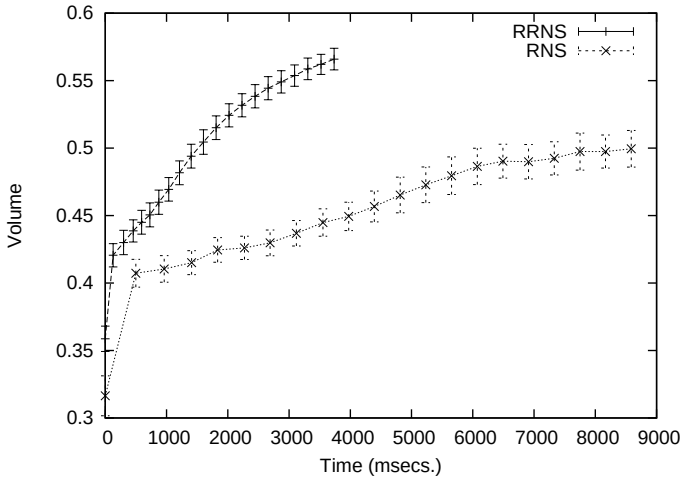


Figure 6.14: Evolution of the non-self covered volume against the time for RNS and RRNS algorithms.

Clearly, the RRNS employs less time on each iteration. This makes more evident the advantage that the new algorithm has in terms of maximizing the covering of the non-self space.

6.5 Summary

This chapter presented a new algorithm to generate antibodies in the non-self space: the Randomized Real-Valued Negative Selection (RRNS) algorithm (Figure 6.1). The algorithm is based on Monte Carlo simulation techniques; this gives it the appellative of *randomized*. The algorithm improves the RNS algorithm proposed in the previous chapter by providing a mathematical support that facilitates:

- the production of a good estimate of the number of antibodies of a given radius needed to cover the non-self space, and
- the provision of a guarantee, at least theoretically, that the algorithm will converge to an optimal configuration.

The RRNS algorithm has two main parts:

- the initial antibody set generation algorithm (function CALCULATE-INIT-ANTIBODY-SET, Figure 6.4), which uses Monte Carlo integration to estimate the number of antibodies needed to cover the non-self set, and

- the antibody distribution optimization algorithm (function OPTIMIZE-ANTIBODY-DISTRIBUTION, Figure 6.9), which uses simulated annealing (also a Monte Carlo method) to optimize the covering of the non-self space.

The second part minimizes a cost function that represents the amount of inter-antibody overlapping. The main assumption is that minimizing the overlapping is equivalent to maximize the volume covered by the set of antibodies. In Section 6.4, this assumption was validated experimentally.

The RRNS algorithm is clearly better than the RNS algorithm in terms of theoretical support. However, this does not mean that it has to be better in terms of performance. In some cases, heuristic algorithms outperform other algorithms with better theoretical support. This is not the case with the RRNS algorithm; the experiments showed that it outperforms the RNS, since it produces a better covering of the non-self space with the same, or less, computational effort.

In conclusion, the algorithm designed in this chapter, RRNS, represents an advance with respect to the original RNS algorithm in terms of theoretical support and performance.

Chapter 7

Conclusions and Future Work

The initial goal that motivated the work in this dissertation was to improve the performance of the NS algorithm and to extend its range of applicability. As was shown in Chapter 3, the binary representation used by current implementations of the NS algorithm has limitations. The following are some of the main limitations:

- Binary matching rules are not able to capture the semantics of some complex self/non-self spaces. This prevents the NS algorithm from producing a good generalization of the non-self space.
- The low-level binary detector representation prevents the extraction of meaningful domain knowledge. This makes it difficult to analyze reasons for reporting an anomaly.
- For some problems, a large number of detectors could be needed to guarantee a good level of detection. This represents a scalability issue.
- The binary representation makes it difficult to integrate the NS algorithm with other immune algorithms, which use higher-level representations (such as real-valued vectors).
- The binary representation provides a crisp distinction between normal and abnormal. Many problems require a 'softer' distinction; that is, the output of the

anomaly detection system must be a degree of abnormality rather than a binary normal-abnormal output.

This research effort concentrated on exploring a new representation for the self/non-self space and algorithms that can overcome these limitations. We chose to use a multi-dimensional real-valued representation of the self/non-self space. The main reasons to do so were as follows:

- it has enough expressive power to represent data on a wide range of problems,
- it is possible to exploit the structure of the $\mathbb{R}^n$ metric space to design efficient detector generation algorithms, and
- this representation is compatible with the representation used by other AIS approaches such as those based on idiotypic immune network theory, which open up scopes for integration.

7.1 Main contributions

A new representation of the self/non-self space requires different detectors (antibodies) and matching schemes. Different options are explored :

- hypercubes, which can be interpreted as an anomaly detection rules;
- fuzzy rules, which are a generalization of the crisp rules defined by the hypercube representation; and
- hyper-sphere-shaped detectors.

The new types of detectors require new algorithms to generate them. We designed and implemented different alternatives:

- Hypercube detectors

  - Negative Selection with Detection Rules (NSDR) using sequential niching (Section 4.2). This is a genetic algorithm that uses a sequential niching technique to evolve a set of detectors (rules) that cover the non-self space.
  - Negative Selection with Detection Rules (NSDR) using deterministic crowding (Section 4.4). This algorithm improves the performance of the previous one by using a different niching technique, deterministic crowding.

- Fuzzy rules
  - Negative Selection with Fuzzy Detection Rules (NSFDR) (Section 4.5). This algorithm uses a similar strategy to NSDR using a deterministic crowding algorithm to generate fuzzy rule detectors.

- Hyper-spheres
  - Real-valued Negative Selection (RNS) (Section 5.4). The algorithm applies a heuristic process that changes iteratively the position of the detectors driven by two goals: to maximize the coverage of the non-self subspace and to minimize the coverage of the self samples.
  - Randomized Real-valued Negative Selection (RRNS) (Chapter 6). This algorithm has a good mathematical foundation that solves some of the drawbacks of the RNS algorithm. Specifically, it can produce a good estimate of the optimal number of detectors needed to cover the non-self space, and the maximization of the non-self coverage is done through an optimization algorithm with proved convergence properties.
  - Additionally, we proposed a hybrid immune learning algorithm that combines the RNS algorithm with conventional classification algorithms to perform anomaly detection (Section 5.3). This method does not use positive or negative detection; rather, it tries to find a boundary between normal and abnormal classes. One of the advantages of this approach is

that it permits the use of a supervised technique for a task that traditionally requires an unsupervised method (as was discussed in Subsection 5.2.1).

We performed multiple experiments applying the proposed algorithms to different data sets. In general, the results showed that the new representation has advantages over the binary representation approach, and its performance is competitive with other anomaly detection techniques. The following are some of the main advantages of this representation:

- It allows the NS algorithm to generalize, that is, to induce the structure of the self set using only a subset of normal samples.

  Chapter 3 showed that binary matching rules cannot capture the structure of a complex self set. Experiments in Sections 4.6 and 5.6 showed that real-valued NS algorithms (NSFDR and RNS) are able to produce a good detection rate while keeping a low false alarm rate, outperforming binary negative selection, even if only a subset of the self set is available for training.

- It facilitates the process of extracting high-level knowledge from normal data.

  The low-level binary representation is not able, in many cases, to represent high-level knowledge of the problem space. The detector representation schemes proposed in this work, such as crisp if-then rules in the NSDR algorithm (Section 4.2) or fuzzy if-then rules in the NSFDR algorithm (Section 4.5), provide a natural way to represent this high-level knowledge. Also, it is possible to use the hybrid immune learning algorithm proposed in Section 5.3 to extract that knowledge by combining the RNS algorithm with a conventional classification algorithm such as a fuzzy rule evolver (Section 5.5) or a neural network (Section 5.6).

- It can produce a non-crisp distinction between self and non-self.

The fuzzy rules produced by the NSFDR algorithm (Section 4.5) or the hybrid immune learning algorithm (Section 5.5) represent a fuzzy characterization of the non-self set. Also, the hybrid immune learning algorithm can produce a non-crisp characterization by using an appropriate classification technique; for instance, in Section 5.6, a neural network was used. This type of network can generate a continuous output in the range [0,1] that assesses the level of abnormality of a sample. This non-crisp distinction between self and non-self facilitates the construction of more flexible anomaly detection systems that can be tuned up to increase or to decrease their sensitivity without re-training.

- It makes the NS algorithm to scale better.

  Our experiments with one high-dimensional data set (Subsections 4.6.2 and 5.6.3) showed that it is possible to use the NS selection algorithm in problems where it is unfeasible to apply the binary version of the NS algorithm. Even though these are preliminary results, they constitute an incentive to explore more general detector representation that can overcome the evident scalability issues of the binary NS algorithm.

7.2 Future work

Our experiments showed that the proposed algorithms outperform binary NS on different aspects. Also, they showed that they can produce results that are comparable with those produced by other anomaly detection strategies based on positive detection. However, it is necessary to perform a more extensive experimentation using different types of data sets and applying other anomaly detection methods in order to assess the real strength of the proposed algorithms.

The hybrid immune learning algorithm proposed in Section 5.3 uses a conventional classification algorithm. In our experiments, we tested it with a genetic-based fuzzy rule generation algorithm (Section 5.5) and with a multi-layer perceptron trained

with back-propagation (Section 5.6). There is a great variety of classification methods; hence, it would be interesting to test additional classification methods in order to determine which ones produce better results when combined with this hybrid approach.

The algorithm proposed in Chapter 6 (RRNS) provides a good mathematical framework that can serve as a basis for the development of better antibody generation algorithms. Specifically, the efficiency of the initial antibody set generation algorithm (Section 6.2) may be improved by using a *Quasi-Monte Carlo integration method* [135], which can reduce the number of iterations needed to produce a good estimate of the non-self volume. These methods replace the random numbers used by Monte Carlo integration with more uniformly distributed deterministic sequences that are known as low-discrepancy sequences (LDS). The use of LDS may produce a better, more uniform initial antibody distribution. Another possibility for improvement is to use a better optimization algorithm for distributing the antibodies in the non-self space. Currently, we are using simulated annealing, which has some interesting theoretical properties; however, there are more efficient optimization algorithms that can be used instead.

The natural immune system combines multiple strategies and mechanisms in order to defend the body against antigens: anomaly detection, clustering, pattern classification, dynamic learning, associative memory, etc. (see Subsection 2.1.2). Most of the current immune-inspired techniques use only a partial subset of immunological principles (see Section 2.2). The integration of different mechanisms in a unified model may provide a better problem solving capability through the synergetic interaction between components. For instance, an AIS that combines self/non-self discrimination (anomaly detection) and idiotypic networks (immune memory) could have very interesting characteristics that are not exhbited by AISs that implement only one of these mechanisms. Such an AIS has not been implemented yet, and one of the reasons is that the AISs that implement self/non-self discrimination (efficiently) use a binary representation that is not compatible with the real-valued representation used by AIS

inspired on idiotypic networks. In this dissertation, we have shown that it is possible (and desireable) to use a real-valued representation to implement an efficient system that performs self/non-self discrimination. Therefore, an interesting extension of this work is to combine the algorithms proposed in this dissertation with some of the current idiotypic-network-based AISs [18, 20, 57, 64].

References

[1] F. Varela, A. Coutinho, B. Dupire, and N. Vaz, "Cognitive networks: immune and neural and otherwise," *Theoretical Immunology: Part Two, SFI Studies in the science of Complexity*, pp. 359–371, 1988.

[2] J. Kuby, *Immunology*. W. H. Freeman and Co., 3 ed., 1997.

[3] C. A. Janeway, "How the immune system recognizes invaders," *Scientific American*, vol. 269, no. 3, pp. 72–79, 1993.

[4] D. Dasgupta, "An overview of artificial immune systems and their applications.," in *Artificial immune systems and their applications* (D. Dasgupta, ed.), pp. 3–23, Springer-Verlag, Inc., 1999.

[5] A. Coutinho, "The self non-self discrimination and the nature and acquisition of the antibody repertoire," *Annals of Immunology. (Inst. Past.)*, vol. 131D, 1980.

[6] J. D. Farmer, N. H. Packard, and A. S. Perelson, "The immune system, adaptation, and machine learning," *Physica D*, vol. 22, pp. 187–204, 1986.

[7] N. K. Jerne, "Towards a network theory of the immune system," *Ann. Immunol. (Inst. Pasteur)*, vol. 125C, pp. 373–389, 1974.

[8] D. E. Cooke and J. E. Hunt, "Recognizing promoter sequences using an artificial immune system," in *Proceedings of the Intelligent Systems in Molecular Biology Conference (ISMB'95)*, pp. 89–97, AAAI Press, 1995.

[9] J. Timmis, *Artificial immune systems: a novel data analysis technique inspired by the immune network theory.* Ph. d thesis, University of Wales, 2000.

[10] 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), pp. 202–212, IEEE Computer Society Press, 1994.

[11] D. Dasgupta and S. Forrest, "An anomaly detection algorithm inspired by the immune system," in *Artificial immune systems and their applications,* (D. Dasgupta, ed.), pp. 262–277, New York: Springer-Verlag, 1999.

[12] S. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary Computation*, vol. 8, no. 4, pp. 443–473, 2000.

[13] D. Dasgupta and S. Forrest, "Tool breakage detection in milling operations using a negative-selection algorithm," Technical Report CS95-5, Department of Computer Science, University of New Mexico, 1995.

[14] 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), pp. 129–135, 1999.

[15] 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* (J. F. C. Harris, ed.), (Cary, NC), pp. 82–87, ISCA, June 1996.

[16] C. A. Coello Coello and N. Cruz Cortés, "A parallel implementation of the artificial immune system to handle constraints in genetic algorithms: preliminary results," in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002* (D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), (USA), pp. 819–824, 2002.

[17] F. Gonzalez, D. Dasgupta, and J. Gomez, "The effect of binary matching rules in negative selection," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, July 2003.

[18] L. N. de Castro and F. J. Von Zuben, "An evolutionary immune network for data clustering.," *Brazilian Symposium on Artificial Neural Networks (IEEE SBRN'00)*, pp. 84–89, 2000.

[19] L. N. de Castro and F. J. V. Zuben, "The clonal selection algorithm with engineering applications.," in *Proceedings of GECCO'00 (workshop proceedings)*, pp. 36–37, 2000.

[20] O. Nasraoui, F. González, and D. Dasgupta, "The fuzzy artificial immune system: Motivations, basic concepts, and application to clustering and web profiling," in *IEEE International Conference on Fuzzy Systems*, (USA), pp. 711–716, IEEE Press, May 2002.

[21] D. Dagupta and F. González, "An immunity-based technique to characterize intrusions in computer networks," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 281–291, June 2002.

[22] F. González and D. Dasgupta, "An imunogenetic technique to detect anomalies in network traffic," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, eds.), (San Francisco, CA), pp. 1081–1088, Morgan Kaufmann Publishers, July 2002.

[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] F. González, D. Dasgupta, and R. Kozma, "Combining negative selection and classification techniques for anomaly detection," in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002* (D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), (USA), pp. 705–710, IEEE Press, May 2002.

[25] F. González and D. Dagupta, "Neuro-immune and self-organizing map approaches to anomaly detection: A comparison," in *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)* (J. Timmis and P. J. Bentley, eds.), (Canterbury, UK), pp. 203–211, University of Kent at Canterbury Printing Unit, Sept. 2002.

[26] F. Gonzalez and D. Dasgupta, "Anomaly detection using real-valued negative selection," *Genetic Programming and Evolvable Machines*, 2003. to be published.

[27] C. A. Janeway, P. Travers, S. Hunt, and M. Walport, *Immunobiology: the immune system in health and disease*. Garland Pub., 1997.

[28] 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. Segel, eds.), New York: Oxford University Press, 2000.

[29] H. Baumann and J. Gauldie, "The acute response," *Immunol. Today*, vol. 15, no. 74, 1994.

[30] I. Roitt, "Specific acquired immunity," in *Essential immunology*, pp. 22–39, Blackwell Science, 9 ed., 1997.

[31] I. Tizzard, "The response of B-cells to antigen," in *Immunology: an introduction*, pp. 199–223, Pub. Saunders College Publishing, 2nd edition ed., 1988.

[32] 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.

[33] J. Kappler, N. Roehm, and P. Marrack, "T Cell Tolerance by Clonal Elimination in the Thymus," *Cell*, no. 49, pp. 273–280, 1987.

[34] 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.

[35] N. K. Jerne, "Clonal selection in a lymphocyte network," in *Cellular selection and regulation in the immune response*, pp. 39–48, Raven Press., 1974.

[36] 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.

[37] D. Dasgupta, *Artificial immune systems and their applications*. New York: Springer-Verlag, 1999.

[38] H. Bersini, "The endogenous double plasticity of the immune network and the inspiration to be drawn for engineering artifacts," in *Artificial immune systems and their applications* (D. Dasgupta, ed.), pp. 22–41, Springer-Verlag, Inc., 2nd edition ed., 1999.

[39] H. Bersini and F. Varela, "Hints for adaptive problem solving gleaned from immune network," in *Parallel problem solving from nature* (H. Schwefel and H. M'hlenbein, eds.), pp. 343 – 354, Springer-Verlag, 1990.

[40] H. Bersini and F. Varela, "The immune recruitment mechanism: A selective evolutionary strategy," in *Proceedings of the 4th International Conference on Genetic Algorithms* (R. Belew and L. Booker, eds.), pp. 520–526, Morgan Kaufman, 1991.

[41] S. Forrest, B. Javornik, R. E. Smith, and A. S. Perelson, "Using genetic algorithms to explore pattern recognition in the immune system," *Evolutionary Computation*, vol. 1, no. 3, pp. 191–211, 1993.

[42] J. O. Kephart, "A biologically inspired immune system for computers," in *Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems ArtificialLifeIV* (R. A. Brooks and P. Maes, eds.), (Cambridge, MA, USA), pp. 130–139, MIT Press, July 1994.

[43] A. Ishiguro, T. Kondo, Y. Watanabe, and Y. Uchikawa, "Dynamic behavior arbitration of autonomous mobile robots using immune networks," in *ICEC'95*, vol. 2, pp. 722–727, 1995.

[44] 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* (J. McHugh and G. Dinolt, eds.), (USA), pp. 110–119, IEEE Press, 1996.

[45] Y. Ishida, "An immune network approach to sensor-based diagnosis for self organization," *Complex Systems*, vol. 10, pp. 73–90, 1996.

[46] P. Hajela, J. Yoo, and J. Lee, "GA based simulation of immune networks - applications in structural optimization," *Journal of Engineering Optimization*, no. 29, pp. 131–149, 1997.

[47] J. Hunt, J. Timmis, D. Cooke, M. Neal, and C. King, "Jisys: The development of an artificial immune system for real world applications," in *Applications of Artificial Immune Systems* (D. Dasgupta, ed.), pp. 157–186, Springer-Verlag, 1999.

[48] D. Dasgupta, "Immunity-based intrusion detection system: a general framework," in *Proceedings of the 22nd national information systems security conference (NISSC)*, pp. 147–160, Oct. 1999.

[49] D. Dasgupta and Y. Cao, "An immunogenetic approach to spectra recognition," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar,

M. Jakiela, and R. E. Smith, eds.), (San Francisco, CA), pp. 149–155, Morgan Kaufmann, 1999.

[50] P. D. Williams, K. P. Anchor, J. L. Bebo, G. H. Gunsch, and G. D. Lamont, "CDIS: Towards a computer immune system for detecting network intrusions," *Lecture Notes in Computer Science*, vol. 2212, pp. 117–133, 2001.

[51] A. Tarakanov and D. Dasgupta, "A formal immune system," in *Third International Workshop on Information Processing in Cells and Tissues (IPCAT'99)*, (Indianapolis), 1999.

[52] D. Bradley and A. Tyrrell, "Immunotronics: Hardware fault tolerance inspired by the immune system," in *Proceedings of the 3rd International Conference on Evolvable Systems (ICES2000)*, vol. 1801, pp. 11–20, Springer-Verlag, Inc., 2000.

[53] L. N. de Castro and F. J. Von Zuben, "An immunological approach to initialize feed forward neural network weights," in *International Conference on Artificial Neural Networks and Genetic Algorithms(ICANNGA'01)*, pp. 126–129, 2001.

[54] A. Tarakanov and D. Dasgupta, "An immunochip architecture and its emulation," in *2002 NASA/DoD Conference on Evolvable Hardware*, 2002.

[55] E. Hart and P. Ross, "Exploiting the analogy between immunology and sparse distributed memories: A system for clustering non-stationary data," in *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)* (J. Timmis and P. J. Bentley, eds.), (Canterbury, UK), pp. 49–58, University of Kent at Canterbury Printing Unit, September 2002.

[56] 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 CEC2002* (D. B. Fogel, M. A.

El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), (USA), pp. 1015–1020, IEEE press, May 2002.

[57] O. Nasraoui, F. González, C. Cardona, C. Rojas, and D. Dasgupta, "A scalable artificial immune system model for dynamic unsupervised learning," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2003. to be published.

[58] A. Somayaji, S. Hofmeyr, and S. Forrest, "Principles of a computer immune system," in *Proceedings of the Second New Security Paradigms Workshop*, pp. 75–82, 1997.

[59] R. J. De Boer and A. S. Perelson, "How diverse should the immune system be?," in *Proceedings of the Royal Society London B*, vol. 252, (London), 1993.

[60] 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)* (J. Timmis and P. J. Bentley, eds.), (Canterbury, UK), pp. 89–98, University of Kent at Canterbury Printing Unit, Sept. 2002.

[61] 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)* (W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, eds.), (San Francisco, CA), pp. 3–10, Morgan Kaufmann Publishers, 9-13 July 2002.

[62] P. Harmer, G. Williams, P.D.and Gnusch, and G. Lamont, "An Artificial Immune System Architecture for Computer Security Applications," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 252–280, June 2002.

[63] L. N. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Approach*. London, UK: Springer-Verlag, 2002.

[64] J. Timmis and M. J. Neal, "A resource limited artificial immune system for data analysis," in *Research and development in intelligent systems XVII, proceedings of ES2000*, (Cambridge, UK), pp. 19–32, 2000.

[65] O. Nasraoui, D. Dasgupta, and F. González, "An novel artificial immune system approach to robust data mining," in *Late Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO)*, (New York), pp. 356–363, AAAI, July 2002.

[66] O. Nasraoui, D. Dasgupta, and F. González, "The promise and challenges of artificial immune system based web usage mining: Preliminary results," in *Proceedings of the Workshop on Web Analytics, Second SIAM Conference on Data Mining*, (Arlington, VA), Apr. 2002.

[67] D. Bradley and A. Tyrrell, "Immunotronics: Novel finite-state-machine architectures with built-in self-test using self-nonself differentiation," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 227–238, June 2002.

[68] C. A. Coello Coello and N. Cruz Cortés, "An approach to solve multiobjective optimization problems based on an artificial immune system," in *First International Conference on Artificial Immune Systems (ICARIS)* (J. Timmis and P. J. Bentley, eds.), (Canterbury,UK), pp. 212–221, University of Kent at Canterbury Printing Unit, Sept. 2002.

[69] D. Dasgupta, "Artificial neural networks and artificial immune systems: Similarities and differences.," in *IEEE International Conference on Systems, Man and Cybernetics*, (Orlando, FL), pp. 873–878, 1997.

[70] L. N. De Castro and F. J. Von Zuben, "Immune and neural network models: Theoretical and empirical comparisons," *International Journal of Com-*

putational Intelligence and Applications (IJCIA), vol. 11, no. 6, pp. 523–535, 2001.

[71] L. N. De Castro and F. J. Von Zuben, "Automatic determination of radial basis function: An immunity-based approach.," *International Journal of Neural Systems (IJNS)*, vol. 1, no. 3, pp. 239–251, 2001.

[72] D. Hawkins, *Identification of Outliers*. London: Chapman and Hall, 1980.

[73] V. Barnett and T. Lewis, *Outliers in Statistical Data*. New York: Wiley, 3rd ed., 1994.

[74] F. Hampel, E. Ronchetti, P. Rousseuw, and W. Stahel, *Robust Statistics*. Wiley, 1986.

[75] P. Huber, *Robust Statistics*. Wiley, 1981.

[76] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.

[77] E. Eskin, "Anomaly detection over noisy data using learned probability distributions," in *Proc. 17th international conf. on machine learning*, pp. 255–262, Morgan Kaufman, 2000.

[78] J. Cannnady, "Next generation intrusion detection: autonomous reinforcement learning of network attacks," in *23th. National Information Systems Security Conference*, pp. 1–12, 2000.

[79] M. Crosbie and E. Spafford, "Applying genetic programming to intrusion detection," in *Working Notes for the AAAI Symposium on Genetic Programming* (E. Siegel and J. Koza, eds.), (MIT, Cambridge, MA, USA), pp. 1–8, AAAI, 10–12 1995.

[80] J. Frank, "Artificial intelligence and intrusion detection: current and future directions," in *Proceedings of the 17th National Computer Security Conference*, Oct. 1994.

[81] R. Kozma, M. Kitamura, M. Sakuma, and Y. Yokoyama, "Anomaly detection by neural network models and statistical time series analysis," in *Proceedings of IEEE International Conference on Neural Networks*, (Orlando, FL), pp. 27–29, 1994.

[82] T. Lane and C. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *ACM Transactions on Information and System Security*, no. 2, pp. 295–331, 1999.

[83] T. Lane, "Hidden markov models for human/computer interface modeling," in *IJCAI-99 Workshop on Learning About Users*, pp. 35–44, 1999.

[84] T. Lane, *Machine learning techniques for the computer security*. PhD thesis, Purdue University, 2000.

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

[86] W. Lee and S. Stolfo, "Data mining approaches for intrusion detection," in *Proceedings of the 7th USENIX security symposium*, (Berkeley, CA), pp. 79–94, USENIX Association, 1998.

[87] W. W. Cohen, "Fast effective rule induction," in *Proc. of the 12th International Conference on Machine Learning*, (Tahoe City, CA), pp. 115–123, Morgan Kaufmann, July 1995.

[88] H. Mannila and H. Toivonen, "Discovering generalized episodes using minimal occurrences," in *Proceedings of the 2nd Intl. Conf. in Knowledge Discovery and Data Mining KDD'96*, pp. 146–151, 1996.

[89] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, (New York), pp. 487–499, Morgan Kaufmann, 1994.

[90] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data," in *Data Mining for Security Applications*, Kluwer, 2002.

[91] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *Proceedings of ACM CCS Workshop on Data Mining Applied to Security*, (USA), ACM Press, Nov. 2001.

[92] 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)* (L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, eds.), (San Francisco, CA), pp. 1330–1337, Morgan Kaufmann, 2001.

[93] J. Balthrop, S. Forrest, and M. R. Glickman, "Revisting LISYS: Parameters and normal behavior," in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002* (D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), (USA), pp. 1045–1050, IEEE Press, 2002.

[94] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard, *Induction: Processes of Inference, Learning, and Discovery*. Cambridge: MIT Press, 1986.

[95] S. Singh, "Anomaly detection using negative selection based on the r-contiguous matching rule," in *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)* (J. Timmis and P. J. Bentley, eds.), (Canterbury, UK), pp. 99–106, University of Kent at Canterbury Printing Unit, sep 2002.

[96] D. Dagupta and F. González, "Evolving complex fuzzy classifier rules using a linear genetic representation," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* (L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, eds.), (San Francisco, CA), pp. 299–305, Morgan Kaufmann, July 2001.

[97] D. Dagupta and F. González, "Information assuarance in computer networks," ch. An intelligent decision support system for intrusion detection and response, pp. 1–14, Springer-Verlag, Inc., 2001.

[98] D. Dasgupta and Z. Michalewicz, *Evolutionary algorithms in engineering applications*. New York: Springer-Verlag, Inc., 1997.

[99] D. Beasley, D. Bull, and R. Martin, "A sequential niche technique for multimodal function optimization," *Evolutionary Computation*, vol. 1, no. 2, pp. 101–125, 1993.

[100] "1999 Darpa intrusion detection evaluation." MIT Lincoln Labs, 1999.

[101] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[102] J. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[103] J. Bentley, "K-D trees for semidynamic point sets," in *Proc. 6th Annu. ACM Sympos. Comput. Geom*, pp. 187–197, 1990.

[104] 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.

[105] D. Mount and S. Arya, "ANN: a library for approximate nearest neighbor searching," in *2nd Annual CGC Workshop on Computational Geometry*, 1997.

[106] 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* (J. Shavlik, ed.), (San Francisco, CA), pp. 445–453, Morgan Kaufmann, 1998.

[107] F. Esponda, S. Forrest, and P. Helman, "A formal framework for positive and negative detection schemes." Draft version, July 2002.

[108] S. W. Mahfoud, "Crowding and preselection revisited," in *Parallel problem solving from nature 2* (R. Männer and B. Manderick, eds.), (Amsterdam), pp. 27–36, North-Holland, 1992.

[109] M. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, 1977.

[110] T. Caudell and D. Newman, "An adaptive resonance architecture to define normality and detect novelties in time series and databases," in *IEEE World Congress on Neural Networks*, (Portland, OR), pp. 166–176, 1993.

[111] 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 '02)* (O. R. Zaïane, R. Goebel, D. Hand, D. Keim, and R. Ng, eds.), (USA), pp. 550–556, ACM Press, 2002.

[112] P. Murphy and D. Aha, "UCI Repository of machine learning databases," 1992.

[113] J. Gomez and D. Dasgupta, "Evolving Fuzzy Classifiers for Intrusion Detection," in *Proceedings of the 2002 IEEE Workshop on Information Assurance*, pp. 68–75, June 2002.

[114] W. Fan, W. Lee, M. Miller, S. Stolfo, and P. Chan, "Using artificial anomalies to detect unknown and known network intrusions," in *Proceedings of the 1st IEEE*

International conference on Data Mining (N. Cercone, T. Y. Lin, and X. Wu, eds.), (USA), pp. 123–130, IEEE Computer Society Press, 2001.

[115] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[116] T. M. Mitchell, "Version spaces: A candidate elimination approach to rule learning," in *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, (Cambridge, MA), pp. 305–310, William Kaufmann, Aug. 1977.

[117] F. Esponda and S. Forrest, "Detector coverage under the r-contiguous bits matching rule," Tech. Rep. TR-CS-2002-03, Department of Computer Science, University of New Mexico, 2002.

[118] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[119] S. Haykin, *Neural networks : a comprehensive foundation*. New York: Macmillan, 1994.

[120] T. Kohonen, *Self-Organizing Maps*, vol. 30 of *Springer Series in Information Sciences*. Berlin, Heidelberg: Springer, 1995. (Second Extended Edition 1997).

[121] D. Dasgupta and N. S. Majumdar, "Anomaly detection in multidimensional data using negative selection algorithm," in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002)* (D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), (USA), pp. 1039–1044, IEEE Press, 2002.

[122] W. H. Wolberg and O. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," *Proceedings of the National Academy of Sciences, U.S.A.*, vol. 87, pp. 9193–9196, Dec. 1990.

[123] "Computational Science Education project. Introduction to monte carlo methods." Oak Ridge National Laboratory, 1995.

[124] J. S. Liu, *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, 2001.

[125] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science, Number 4598, 13 May 1983*, vol. 220, no. 4598, pp. 671–680, 1983.

[126] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[127] G. S. Fishman, *Monte Carlo. Concepts, algorithms, and Applications*. Springer-Verlag, 1996.

[128] H. D. Brunk, *An Introduction to Mathematical Statistics*. Blaisdell Publishing Co., 1965.

[129] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pp. 322–331, ACM Press, 1990.

[130] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *Proceedings of the 10th ACM International Conference on Management of Data (SIGMOD)* (S. B. Navathe, ed.), pp. 47–57, ACM Press, 1985.

[131] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[132] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice.* Chapman_Hall, 1996.

[133] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; part 1, graph partitioning," *Operations Research*, vol. 37, pp. 865–892, Nov. 1989.

[134] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Computing Surveys (CSUR)*, vol. 30, no. 2, pp. 170–231, 1998.

[135] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

Summary of Data Sets Used for Experiments

1. Data set name: Mackey-Glass

    - Number of features: 4
    - Training set size: 497
    - Testing set size: 497
    - Description: Chaotic time series generated by a differential equation.
    - Source: The differential equation is solved numerically using the fourth-order Runge-Kutta method. The features are extracted by sliding a window (of size four) through the time series (see Subsection 4.6.1.1)

2. Data set name: IRIS

    - Number of features: 4
    - Training set size: 40
    - Testing set size: 110
    - Description: Three different type of flowers described by four features.
    - Source: Initially created by Fisher [118]. This particular version of the data set was obtained from the University of California Machine Learning Repository [112] (ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris).

3. Data set name: MIT-Darpa 98

    - Number of features: 32
    - Training set size: 1,474
    - Testing set size: 415,978
    - Description: Network traffic data generated in a controlled environment.
    - Source: Data set used for the KDD Cup 99 competition available at the University of California Machine Learning Repository [112] (`http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`).

4. Data set name: MIT-Darpa 99

    - Number of features: 9
    - Training set size: 4,000
    - Testing set size: 5,192
    - Description: Network traffic data generated in a controlled environment.
    - Source: The original tcpdumd data was obtained from MIT Lincoln Lab [100]. It was processed to extract *nine* traffic features (see Subsection 4.3.1).

5. Data set name: Wisconsin Breast Cancer

    - Number of features: 9
    - Training set size: 271
    - Testing set size: 412
    - Description: The features describe characteristics of the cell nuclei present in the digitized image of a fine needle aspirate of a breast mass.
    - Source: Created at the University of Wisconsin Hospitals [122]. This particular data set was obtained from the University of California Machine

Learning repository [112]
(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin).