2005

A Self-Adaptive Evolutionary Negative Selection Approach for Anomaly Detection

Luis J. Gonzalez
Nova Southeastern University, lujogre@gmail.com

A Self-Adaptive Evolutionary Negative Selection Approach for Anomaly Detection

by

Luis J. Gonzalez

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Graduate School of Computer and Information Sciences
Nova Southeastern University

2005

We hereby certify that this dissertation, submitted by Luis J. Gonzalez, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

James Cannady, Ph.D. Chairperson of Dissertation Committee	Date
David Fogel, Ph.D. Dissertation Committee Member	Date
Sumitra Mukherjee, Ph.D. Dissertation Committee Member	Date
Greg Simco, Ph.D. Dissertation Committee Member	Date

Approved:

Edward Lieblein, Ph.D. Dean, Graduate School of Computer and Information Sciences	Date

Graduate School of Computer and Information Sciences
Nova Southeastern University

2005

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

A Self-Adaptive Evolutionary Negative Selection Approach for Anomaly Detection

By

Luis J. Gonzalez

January 2005

Forrest et al. (1994; 1997) proposed a negative selection algorithm, also termed the exhaustive detector generating algorithm, for various anomaly detection problems. The negative selection algorithm was inspired by the thymic negative selection process that is intrinsic to natural immune systems, consisting of screening and deleting self-reactive T-cells, i.e., those T-cells that recognize self cells.

The negative selection algorithm takes considerable time (exponential to the size of the self data) and produces redundant detectors. This time/size limitation motivated the development of different approaches to generate the set of candidate detectors.

A reasonable way to find suitable parameter settings is to let an evolutionary algorithm determine the settings itself by using self-adaptive techniques.

The objective of the research presented in this dissertation was to analyze, explain, and demonstrate that a novel evolutionary negative selection algorithm for anomaly detection (in non-stationary environments) can generate competent non-redundant detectors with better computational time performance than the NSMutation algorithm when the mutation step size of the detectors is self-adapted.

Acknowledgments

I would like to thank Dr. James Cannady for his support of my research, and Drs. Sumitra Mukherjee, Greg Simco, and David Fogel for their support as dissertation committee members. I would also like to thank Leandro N. de Castro from the Universidade Católica de Santos (Unisantos), Brazil, for his suggestion to explore the use of self-adaptive mechanisms in artificial immune systems and Modupe Ayara and Jonathan Timmis from the University of Kent at Canterbury (UKC) for providing the source code of the NSMutation algorithm and comments during the research.

In addition, I must give a special recognition to my wife Beatriz for her understanding and patience while this research was conducted. My beloved father "Nene" whose teachings have been the guide to my existence. My mother "Mamia" who encouraged me to keep it up, and the supreme energy that supports the universe: God.

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Problem Background

To date, research has found that many of the characteristics of natural immune systems can be applied to solve real-life computer/technical problems (e.g., pattern recognition, fault and anomaly detection, and the security of information systems). Researchers have focused on creating artificial immune systems that can differentiate between *self* and *non-self* states, where *self* can take on different definitions, such as normal network traffic between computers.

The maturation of T-cells in the thymus gland inspired the creation of the negative selection algorithm, also termed the exhaustive detector generating algorithm, to differentiate between *self* and *non-self* states (Forrest, Perelson, Allen, & Cherukuri, 1994). Later, Ayara, Timmis, de Lemos, de Castro, & Duncan (2002b) investigated the usefulness of the exhaustive detector generating algorithm and three variations: linear (D'haeseleer, Forrest, & Helman 1996), greedy (D'haeseleer et al., 1996), and binary template (Wierzchon, 2000). Ayara et al. (2002b) found that:

- The exhaustive detector generating algorithm takes an amount of time that grows exponentially with the size of the self data and produces redundant detectors.
- The linear algorithm has a linear time complexity, but also produces redundant detectors.

- The greedy algorithm produces a complete repertoire of detectors that uses as much space as the linear algorithm, but has a higher computational complexity as compared to the linear algorithm.
- The binary template produces a minimal set of efficient detectors at the expense of more computation time.

Based on these drawbacks, Ayara et al. (2002b) proposed a new algorithm, termed the NSMutation, which was similar to the exhaustive detector generating algorithm except that it (1) was suitable for any matching rule, (2) eliminated redundancy, and (3) possessed parameters (e.g., repertoire size, probability of failure, mutation probability, and detector life-time indicator) that can be optimized manually through parameter tuning for better performance.

Elsewhere, Eiben, Hinterding, and Michalewicz (1999) offered that parameter tuning in evolutionary algorithms based on manual trial and error is impractical because:

1. Parameters are interdependent, and trying all different combinations systematically is practically impossible.
2. The process of parameter tuning is time consuming, even if the parameters are optimized sequentially, regardless of their interactions.
3. For a given problem, the selected parameter values are not necessarily optimal, even if the effort made in setting them is significant.

Eiben et al. (1999) suggested using a parameter control approach as an alternative in order to overcome the above limitations.

The parameter control approach encompasses all the methods used for changing the value of a parameter dynamically, and can be classified into one of three categories:

1. Deterministic parameter control when a deterministic rule alters the values of the strategy parameters without any feedback to approve or reject the change.
2. Adaptive parameter control when some form of feedback determines the direction of the changes to the strategy parameter.
3. Self-adaptive parameter control when the strategy parameters values are encoded into the individual and undergo variation to obtain a competent optimal population of individuals.

When evolutionary algorithms that use self-adaptive parameter control techniques have been applied to real-life problems, the results have shown statistically significantly improved solutions and better computational time performance as compared to evolutionary algorithms that did not use any self-adaptation (Chellapilla & Fogel, 1997; Hinterding, Michalewicz, & Peachey, 1996).

Problem Statement

The quality of the solutions (detectors) generated by the NSMutation is highly dependent on the strategy parameter values. These values are not necessarily optimal when they are calculated manually. Here, quality refers to: (1) non-redundant detectors and (2) detectors that do not bind (match) to *self* with any affinity at all, where *self* encompass the normal network activity between two or more computers. It is desired to optimize the strategy parameters automatically.

Goal

The goal of this research was to analyze, explain, and demonstrate that a novel evolutionary negative selection algorithm for anomaly detection (in non-stationary

environments) that uses self-adaptive control techniques can outperform the NSMutation algorithm within the computational time necessary to generate the minimum repertoire of competent detectors needed to identify anomalies in a given data set of network activity and without sacrificing the quality of the detectors.

Evolutionary algorithms are a class of problem-solving methods inspired by the Darwinian theory on the origin of the species using natural selection (1859). The rationale of the evolutionary algorithm approach relies on the concepts of (1) variation of solutions in a population, (2) competition between those solutions, and (3) selection of those with superior performance.

Strategic parameters control evolutionary algorithms. These parameters determine the type and/or probability of mutation, recombination and other possible variations of the individuals to be evolved. Rules control the variation of these strategy parameters. When the rules are predetermined and specify how to modify the parameters, then the rules are designated as absolute update rules, and the algorithms that use these rules are called adaptive. Elsewhere, when the competitive process inherent in the evolutionary computation is what determines whether the changes in the strategy parameters are advantageous, then the rules are called empirical, and the algorithms that use them are called self-adaptive.

Predicted Advantages of a Self-Adaptive Evolutionary Negative Selection Approach

1. The process of calculating mutation probabilities manually will no longer be required (this process is time consuming and impractical). The self-adaptive approach will replace manual calculation.

2. The novel, self-adaptive evolutionary negative selection algorithm will outperform the NSMutation algorithm in the computational time required to generate detectors but without sacrificing the quality of detectors.
3. Detectors of better quality than those generated by the NSMutation will be obtained. In this particular case, the best individuals (detectors) or solutions will be the individuals (detectors) that will survive at the end of the algorithm (in a manner similar to Darwin's concept of evolution).

Research Questions

The primary research question was to determine the computation time required to generate competent detectors using the self-adaptive evolutionary negative selection algorithm (SANSAD) as compared to the computation time required to generate competent detectors using the NSMutation algorithm. In addition, this research project sought to answer the following questions:

- What is the percentage of true detections of the detectors generated by the self-adaptive evolutionary negative selection algorithm as compared to the percentage of true detections of the detectors generated by the NSMutation algorithm?
- What is the percentage of false detections of the detectors generated by the self-adaptive evolutionary negative selection algorithm as compared to the percentage of false detections of the detectors generated by the NSMutation algorithm?
- Is the novel self-adaptive evolutionary negative selection algorithm more efficient and effective than the NSMutation algorithm?
- When and why is the novel self-adaptive evolutionary negative selection algorithm better than the NSMutation algorithm?

- When and why is the novel self-adaptive evolutionary negative selection algorithm not better than the NSMutation algorithm?

Hypothesis Statement

A novel evolutionary negative selection algorithm for anomaly detection (in non-stationary environments) that uses self-adaptive techniques (dynamic parameter setting) to mutate the mutation step size of the detectors can generate detectors of better quality and with better computational time performance than the NSMutation algorithm.

Relevance

The negative selection algorithm has been used mainly for different aspects of anomaly detection, for example, in computer security, DNA-based negative selection, virus detection and elimination, image inspection systems, image segmentation, and novelty detection in time series data (Dasgupta & Gonzalez, 2002; Esponda, Forrest, & Helman, 2002; Gonzalez & Dasgupta, 2002; Gonzalez & Dasgupta, 2002a; Gonzalez, Dasgupta, & Kozma, 2002; Kim & Bentley, 1999; Kim & Bentley, 2001). Even so, Kim and Bentley (2001) have suggested that the negative selection approach is impractical because generating valid detectors requires an unacceptably long computation time when large data sets are used (e.g., data generated from a network activity). The time required for generating detectors grows exponentially with the size of the self set. Several efforts have been made to optimize the generation of detectors; however, an efficient solution that addresses the significant computation time to generate detectors while maintaining the same quality of detectors has not been found. The computation time required for generating detectors is increased when non-redundant detectors that do not bind to self

with any affinity at all are produced. Elsewhere, the quality of detectors is lower when the computation time is decreased.

Some research (Angeline, 1996; Chellapilla & Fogel, 1997; Fogel, Angeline, & Fogel, 1995; Glickman & Sycara, 1998; Hinterding & Michalewicz, 1996; Spears, 1995) indicated that the use of self-adaptive mechanisms in evolutionary algorithms yielded significant performance improvements from a computation time perspective without any corresponding sacrifice in the quality of the solutions. This evidence suggests that if a novel evolutionary negative selection algorithm can make use of self-adaptive parameters, instead of static parameters, the computational time required to generate detectors using the novel evolutionary negative selection algorithm will be less than the computational time required to generate detectors using NSMutation. Furthermore, the percentage of correct detections will increase, and the percentage of incorrect detections will decrease, as compared to the percentage of detections (correct and incorrect) of the detectors generated by the NSMutation algorithm.

Significance

The existing negative selection algorithms that possess evolutionary features, for example the NSMutation algorithm, require that the value of their strategy parameters be tuned manually, e.g., the mutation probability and the detector lifetime indicator. Michalewicz and Fogel (2000, p. 280) indicate that the labor required to tune the parameters of an evolutionary algorithm is too time consuming and impractical when the strategy parameters are varied by manual trial and error. Michalewicz and Fogel (2000, pp. 299-300) also state that in general a practical way to find suitable parameter settings is to let the evolutionary algorithm determine the settings itself by using self-adaptive

techniques. Algorithms that use self-adaptive techniques have shown significantly improved solutions and better computational time performance statistically as compared to algorithms that do not use any self-adaptation (Chellapilla & Fogel, 1997; Hinterding, Michalewicz, & Peachey, 1996).

The overview described here offers support for the hypothesis that a novel evolutionary algorithm that can use self-adaptive parameters for anomaly detection in non-stationary environments will significantly advance artificial immune systems.

Barriers and Issues

The primary advantage of evolutionary algorithms[1] is their conceptual simplicity (Fogel, 1997). The basic procedure is very straightforward. An evolutionary algorithm generates a population of potential solutions to a problem, uses variation operators that create new solutions from existing ones, and applies a selection mechanism for keeping those solutions that have worked best up to the current moment in time. Although the procedure appears very simple, the research presented here faced the following constraints:

- The mutual influence of different parameters and the combined influences of parameters, when taken together, on the behavior of an evolutionary algorithm, can be very complex.
- The possible solutions are so heavily constrained that constructing even one feasible answer is difficult, let alone searching for an optimum solution.

[1] The term evolutionary algorithm is used in this work to describe any of the algorithms that use population-based random variation and selection.

- There is very little information about which of the adaptive approaches (population, individual, or component level), if any, is most effective in certain circumstances. Although the number of parameters for a population-level adaptive technique is smallest, and consequently easiest to adapt, those parameters may not provide enough control over the evolutionary computation to permit the most efficient processing. This assertation is true because the best way to manipulate an individual may differ significantly from the best way to manipulate the population on average. In contrast, component-level adaptation affords the most adaptive control; however, the number of parameters may inhibit efficient adaptation in certain circumstances.

In summary, the effectiveness of an evolutionary algorithm depends on its components (e.g., representation, variation operators) and their interactions with the environment of action. The variety of parameters included in these components, the many possible choices (e.g., to change or not to change), and the complexity of the interactions between various components and their parameters can make the selection of a "best" evolutionary algorithm for a given problem very difficult.

This research thus required an interdisciplinary synthesis of concepts that ranges from artificial immune systems and problem solving to computer security.

Limitations and Delimitations of the Research

Limitations

One limitation of this study was the inability to randomly sample the entire scope of computer network activity. This study was restricted to the data obtained from the Information Exploration Shootout project (Institute for Visualization and Perception Research, 2001).

Delimitations

The approach of the self-adaptive evolutionary negative selection can be used and implemented in different domains, e.g., pattern recognition, data analysis, and computer security. Attention was restricted here only to the experimentation of the self-adaptive evolutionary negative selection algorithm in the context of computer security, specifically in the identification of network activity patterns that exhibit a behavior different from that of the normal user (anomaly detection).

Definition of Terms

Anomaly detection: A component of the computer's protection mechanisms used to identify a computer intrusion that exhibits a pattern of behavior different from that of the normal user.

Antibody: Protein produced by the immune system of humans and higher animals in response to the presence of a specific antigen.

Antigen: A foreign substance (usually a protein) which, when introduced into the body, stimulates an immune response.

Apoptosis: The process by which a cell kills itself intentionally (programmed cell death) in response to problems within the cell or to signals from outside the cell.

B-cells: A type of cell produced by the bone marrow which, when stimulated by an antigen, becomes either a memory cell or a plasma cell that forms antibodies against the antigen.

Chromosome: A biological structure in a cell nucleus that consists of genes and non-coding DNA. The chromosomes contain the genetic information of a living being. In the case of evolutionary algorithms, individuals that belong to a population of possible solutions are sometimes referred to as chromosomes.

Crossover: From an evolutionary computing perspective, the process of interchange of sections between pairing homologous strings (chromosomes). This process also is known as recombination.

Detector Lifetime Indicator: Defined by Ayara et al. (2002b) as the number of times that mutation can be performed on a random detector.

Evaluation Function: The criteria used to to measure the performance of individuals in a population of solutions.

Evolutionary Algorithms: All the algorithms that use population-based random variation and selection.

False detection: When a detector identifies and reports as incorrect (non-self) something that is correct (self).

Hamming Distance: The minimum number of bits that must be changed in order to convert one bit string into another. This distance can be found by using XOR on corresponding bits or equivalently, by adding corresponding bits (base 2) without a carry.

Lymphocytes: Any one of a group of white blood cells (T-cells and B-cells) of crucial importance to the adaptive part of the body's immune system.

Matching Rule: Criterion used to determine the affinity between two strings. Two strings match if the distance between them is equal to a given matching threshold.

Matching Threshold: Value used to determine whether two strings match or not. This value should not be greater than the string length representing the distance between two strings. For example, if eight is the length of the string representing the distance between String A and String B, then the matching threshold cannot be greater than that number (8) nor less than one (1).

MHC Proteins: Defined by Sompayrac (1999) as proteins encoded by the major histocompatibility complex (the region of a chromosome that includes a complex of genes involved in antigen presentation).

Mutation Probability: Defined by Ayara et al. (2002b) as a threshold that determines whether a bit position in a binary string will be mutated or not. This value is

determined adaptively and is calculated as the length of bits in a random detector that match self, divided by the length l.

Mutation: Defined from an evolutionary computing perspective as the change or alteration in form or qualities of a string or other coding structure.

Negative Selection: According to Sompayrac (1999), the process by which T-cells with receptors that recognize abundant self antigens in the thymus are anergized or deleted. This process is also called central tolerance induction.

Peptides: Small protein fragments or any compound of two or more amino acids.

Probability of Failure: Probability that competent detectors fail to detect non-self.

R-contiguous Distance: The minimum number of contiguous bits that must be changed in order to convert one bit string into another.

Reproduction: Process by which an evolutionary algorithm copies a single individual or combines two or more members of a population to create an individual.

T-cells: A cell produced by the thymus, which participates in the immune system's response to infected or malignant T-cells.

Theoretical Parameter Setting: When the algorithm set the parameters using the mathematical equations provided by Forrest et al. (1994).

Thymus: A gland located in the upper chest that produces T-cells and is fundamental during the early years of existence in the formation of an efficient immune system.

True Detection: When a detector identifies and reports non-self correctly.

Summary of the Chapter

Studies have been conducted to obtain a solution to the time/size limitation of the process of generating the set of candidate detectors required by negative selection, e.g., linear, greedy, binary template, and the NSMutation. Of these approaches, the NSMutation has been demonstrated to be superior (Ayara et al., 2002b), given that it is suitable for any matching rule, generates non-redundant detectors, and has the advantage of possessing tunable parameters. However, the quality of the solutions (detectors) generated by the NSMutation is highly dependent on the strategy parameters values, which are not necessarily optimal when they are determined manually (inaccurate strategy parameter values will cause inaccurate solutions). Elsewhere, Michalewicz and Fogel (2000, pp. 299-300) indicate that a reasonable way to find suitable parameter settings for an evolutionary algorithm is to let that evolutionary algorithm determine the settings by itself by using self-adaptive techniques. Hence, the idea of using self-adaptive techniques to determine the settings of the mutation step size of a negative selection approach was conceived. The reality that the effectiveness of an evolutionary algorithm depends on many of its components was explained within the barriers and issues section. The inability to randomly sample the entire population of computer network activity was mentioned as a limitation of the study. The experimentation of the self-adaptive evolutionary negative selection algorithm only in the context of computer security was stated as a delimitation of the study.

Chapter 2

Review of the Literature

Introduction

The literature review provided here establishes the background for the research. The review begins with an explanation of the immune system, follows with an overview of artificial immune systems, continues with an historical overview of the negative selection algorithm, an outline of evolutionary algorithms and the self-adaptation approach, and concludes with an analysis of the convergence properties of algorithms of the form $(\mu + \lambda) - EA$.

The Nature of the Immune System

The immune system is a network that involves many different T-cells or denominated lymphocytes that protect the body from bacterial, parasitic, fungal, viral infections, and the growth of tumor cells. The two major classes of lymphocytes are B-cells, which grow to maturity in the bone marrow, and T-cells, which mature in the thymus.

B-cells produce antibodies that circulate in the blood and lymph streams and attach to foreign antigens to mark these antigens for destruction by other immune cells. B-cells are part of what is known as antibody-mediated or humoral immunity, so named because the antibodies circulate in the blood and lymph streams.

Some T-cells, which patrol the blood and lymph system looking for foreign invaders and marking antigens, attack and destroy diseased cells that they recognize as

foreign. T lymphocytes are responsible for cell-mediated immunity (or cellular immunity). T-cells also co-participate in the orchestration, regulation, and coordination of the overall immune response. T-cells depend on unique cell surface molecules called the major histocompatibility complex (MHC), which help the T cells to recognize antigen fragments.

Figure 1 illustrates the primary and secondary lymphoid organs. The primary lymphoid organs or sites of lymphocyte maturation in most mammals are bone marrow and thymus; in rabbits - Peyer's Patches and appendix; in sheep and cattle – Peyer's Patches and other gut associated primary lymphoid tissue, in birds – Bursa of Fabricius. The secondary lymphoid organs or sites where lymphocytes encounter antigens and interact with other cells of the immune system are lymph nodes and the spleen.

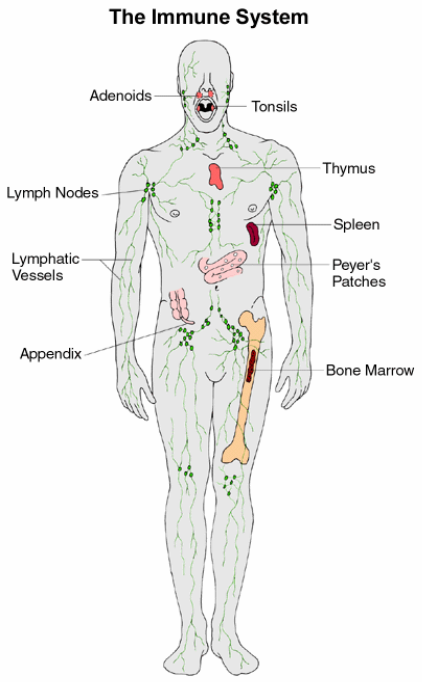


Figure 1. The immune system[2]

[2] From: Muschealth.com – The Medical Resource. Retrieved November 12, 2003, from http://www.muschealth.com/infectious/immune.htm

The Negative Selection Process (Central Tolerance Induction)

The purpose of a negative or down-regulatory signal following certain lymphocyte-antigen interactions is to allow for the control of those lymphocytes bearing anti-self receptors. The negative selection of a lymphocyte is the process by which a lymphocyte-antigen interaction results in the death (or anergy) of that lymphocyte. The T- or B-cell is simply purged from the repertoire. Each organ of the lymphoid system plays a role in this negative selection. The primary lymphoid organs are designed primarily to exclude foreign antigens and preserve the self-antigens; the secondary lymphoid organs are designed to filter out and concentrate foreign material and thus promote stimulatory intercellular immune reactions.

The Negative Selection of T-Cells

The negative selection of T-cells can occur within the thymus (central tolerance) or outside the thymus (peripheral tolerance). Negative thymic selection is based on certain considerations. The thymus is comprised of a myriad of Class I and Class II MHC-bearing APCs, including macrophages, dendritic cells, and specialized epithelial cells. As a blood-thymic barrier protects the thymus, these APCs primarily present self-peptide/MHC complexes to the emerging T-cell repertoire. Negative thymic selection stems from interactions of immature T-cells with the self-peptides presented by the self-MHC molecules on the thymic APC. This process results in the death of an activation-dependent cell, thereby purging potentially autoreactive T-cells from the repertoire. T-cells bearing useless TCRs (T-cells Receptors) that do not exhibit significant interactions with any self-MHC ligands are lost from the repertoire through positive selection. The time and extension of this process of deletion depends upon the affinity (the attraction

between an antigen and an antibody) of the binding between the TCR and the self-antigen. T-cells that bind with higher affinities to the self-antigen are purged more effectively from the repertoire than those that bind with lower affinities.

Although the negative selection nearly eliminates the entirety of developing thymocytes, self-reactive T-cells can still escape from the thymus and circulate in the periphery as fully immuno-competent T-cells. These self-reactive T-cells can pose a threat of an autoimmune disease taking hold of the host. The inductive signal for T-cell activation requires more than the binding of a TCR with a MHC/peptide complex. For the T-cells on the periphery, several adjunct processes, such as the binding of a variety of cell adhesion molecules, are necessary for T-cell activation. In the absence of co-stimulatory activity, the union of a TCR and a MHC/peptide complex may deliver a down-regulatory signal to the T-cell. This innate immunity (defense against infections, but not requiring any previous experience with the antigen or infectious agent) delivers a great amount of co-stimulatory signals that establish an adaptive or acquired immunity (defense against infections developed when the body is exposed to various antigens).

The Negative Selection of B-cells

T-cell tolerance alone would be insufficient protection against autoimmunity. Immature B-cells within the bone marrow are especially sensitive to tolerance induction. Mature B-cells can also be rendered tolerant if they encounter an antigen in the absence of both T-cell help and co-stimulatory influences. As with the T-cells, self-reactive B-cells can also escape the central B-cell negative selection. In this case, B-cell activation or tolerance will be the result of the number, the strength, and the time when the co-stimulatory signals arise. A fast and sudden ligation (characteristic of foreign antigens) of

the receptor to the antigen will generally induce a clonal response. At the same time, a constant and relatively weak stimulation (typical of self-antigens) will lead to tolerance, characterized by the inhibition of the clonal response and further cellular apoptosis (programmed individual cell death that occurs normally in its development, during aging, and in various pathologic conditions).

Summary of the Immune System

The immune system is a network of different kinds of cells that participate in the actual immune response. Of these, one important class of cells is the lymphocytes: B-lymphocytes (produced in the bone marrow) and T-lymphocytes (called T-cells because they mature in the thymus). Lymphocytes are the primary detectors of the immune system. Each individual lymphocyte is specialized so it can respond to a limited group of structurally related antigens by having receptors on their surfaces to detect foreign proteins (called antigens). A pseudo-random genetic process produces these receptors, and it is highly likely that some of the receptors detect self-molecules. T-cells undergo a censoring process in the thymus, called negative selection, in which any T-cells that recognize self are destroyed. T-cells that do not bind to self-peptides, however, leave the thymus and provide the basis for the body's immune protection against foreign antigens. Figure 2 illustrates the positive and negative selection of thymocytes in the thymus.

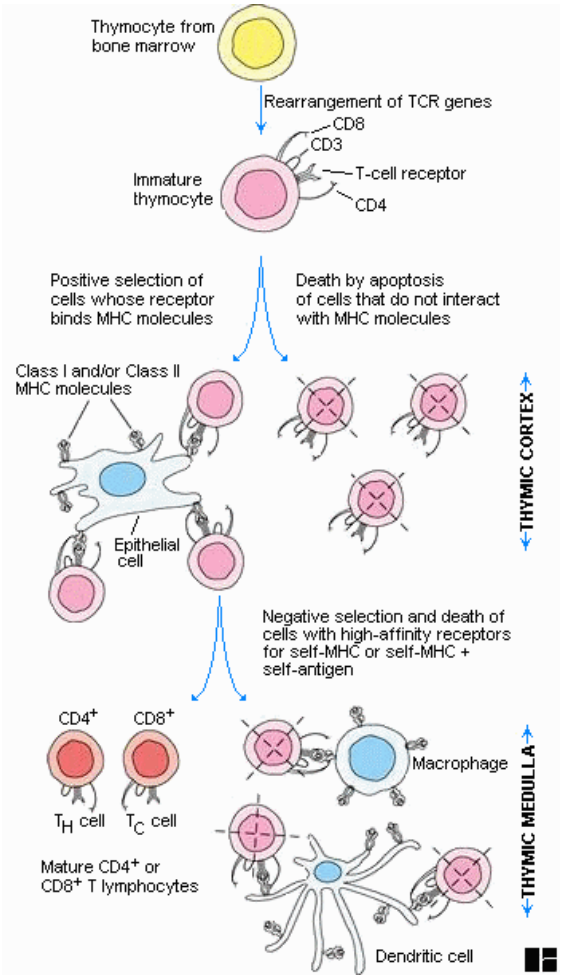


Figure 2. Positive and negative selection of thymocytes in the thymus[3]

Artificial Immune Systems

The implementation of specific functions of the natural immune systems, i.e., the T-cell maturation process and using mathematical representations, has been the basis for the appearance of the artificial immune systems as a new computational paradigm.

So far, many of the functions of the immune system have been modeled algorithmically and have been applied successfully to the solution of real-life problems, e.g., pattern recognition, fault and anomaly detection, data analysis (data mining and

classification), and the security of information systems. Artificial immune systems can thus be defined as the application of specific biological immune functions, modeled previously using mathematical analysis, to real-world problem solving involving computers.

Algorithms of Artificial Immune Systems

The two main sources of the immune system responsible for the generation of immunological cells (the bone marrow and the thymus) and the two basic immunological theories (clonal selection and the immune network), have been the models used for creating artificial immune system algorithms and are detailed as follows.

Bone Marrow Models

The bone marrow is a soft tissue located in the cavities of the bones that is responsible for the generation of all blood cells. Some artificial immune systems have been developed by imitating the bone marrow function, e.g., the generation of a repertoire of strings that can be used for different domains of defense.

Thymus Models

The thymus generates cells that are capable of differentiating between self and non-self. These negative and positive selections are two main processes that participate in the selection and maturation of qualified cells. Both processes have been modeled algorithmically and used for different identification purposes, e.g., the identification of

---

[3] From: Kuby, J. (1998). *Immunology* (3rd). Retrieved November 12, 2003 from http://www.whfreeman.com/immunology/CH12/kuby12.htm

network activity patterns that exhibit a behavior different from that of the normal user (anomaly detection).

Clonal Selection Algorithms

The clonal selection theory states that an antigen will select the more qualified lymphocytes to be proliferated. The level of affinity with the antigen will measure the qualification of the lymphocyte. In other words, the lymphocyte that possesses receptors with the greater capability of reacting with the antigen is the more qualified lymphocyte. Researchers have developed algorithms for pattern recognition, inspired by the clonal selection principle, e.g., the CLONALG (de Castro, and Von Zuben, 2000b)

Immune network models

In contrast to the clonal selection theory, where the grade of affinity of the lymphocytes is measured with respect to an external element (the antigen), immune network theory states that lymphocytes, specifically B-cell receptors (antibodies), have some portions called idiotopes, which make possible the measurement of the grade of affinity between lymphocytes. The immune network model inspired the development of some continuous models, e.g., the model of N.K. Jerne; the model of J.D. Farmer and collaborators; the model of F. Varela and A. Coutinho; and some other discrete models, e.g., the algorithm of J. Timmis and collaborators; the model of L.N. de Castro and F.J. Von Zuben (de Castro, and Timmis, 2002a).

An Historical Overview of the Negative Selection Algorithm

Forrest et al. (1994; 1997) proposed a negative selection algorithm for various anomaly detection problems inspired by the thymic negative selection process that is intrinsic to the natural immune system, which consists of the screening and deletion of self-reactive T-cells or those T-cells that recognize self cells.

The negative selection algorithm defines 'self' by modeling the normal behavior patterns of a monitored system. The algorithm then generates random patterns that are compared to each known self pattern. If any randomly generated pattern matches an existing self pattern, the generated pattern fails to become a detector and is removed. Otherwise, the randomly generated pattern becomes a 'detector' and examines the subsequent profiled patterns of the monitored system. During this monitoring stage, a new anomaly is presumed if a 'detector' matches any newly profiled pattern.

The Negative Selection Algorithm

The negative selection algorithm can be described as a mathematical representation of the maturation of T-cells in the thymus gland and summarized (de Castro and Timmis, 2002) as follows:

1. Initialization: Randomly generate strings and place them in a set P of immature T-cells. Assume all molecules (receptors and self-peptides) are represented as binary strings of the same length L;
2. Affinity evaluation: Determine the affinity of all T-cells in P with all elements of the self set S;
3. Generation of the available repertoire: Indicate that if the affinity of an immature T-cell (element of P) with at least one self-peptide is greater than or equal to a

given cross-reactive threshold ϵ, then the T-cell recognizes this self-peptide and has to be eliminated (negative selection); otherwise the T-cell is introduced into the available repertoire A.

Figure 3 shows a flowchart of the exhaustive detector generating algorithm.

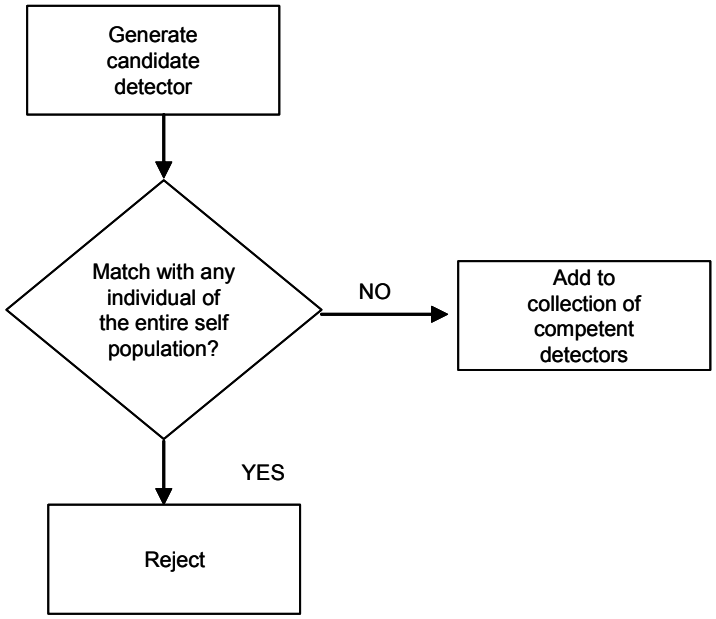


Figure 3. A flowchart of the exhaustive detector generating algorithm

The Drawbacks of the Negative Selection Algorithm

Kim and Bentley (2001) found that the negative selection algorithm, as applied to anomaly detection, is inefficient (from a computational time perspective) when the largest data sets are used (e.g., 600,000 or more records of TCP packet headers or network activity between two or more computers).

The negative selection algorithm is considered too time consuming for large data sets because the number of detectors increases exponentially with the size of the population. This time/size limitation of the algorithm motivated the development of

different approaches to generate the set of candidate detectors: (1) linear (D'haeseleer et al., 1996), which has linear complexity and produces redundant detectors; (2) greedy (D'haeseleer et al., 1996), which produces a complete repertoire of detectors but requires a greater computational time as compared to the linear algorithm; (3) binary template (Wierzchon, 2000), which produces a minimal set of efficient detectors at the expense of more computation time; and (4) NSMutation (Ayara et al., 2002b), which produces non-redundant detectors and possesses tunable parameters (e.g., probability of failure and mutation probability).

Gonzalez and Dasgupta (2002) indicated that generating effective detectors and changing the representation scheme could overcome the severe scaling problems of the negative selection algorithm for handling real traffic data. The approach presented by Gonzalez and Dasgupta (2002), however, does not rely on a structured representation of the data and uses only normal data to build a profile of the system.

The core of the technique then is an evolutionary algorithm that evolves rules to cover the abnormal space (non-self), but without the advantages of self-adaptive techniques for the mutation step size, which can contribute to generate efficient (from a computational time perspective) competent detectors.

Summary of Knowns and Unknowns Regarding the Negative Selection Algorithm

The main observations about the negative selection algorithm include the following:

1. The algorithm is tunable with relation to the probability of detection.
2. The size of a detector does not necessarily grow with the number of strings being protected.

3. Detection is symmetric. The same matching process that notices changes to self, detects changes to the detector set. Hence, there is no a priori way to decide if the change was to self or to the detectors when a change is detected (Forrest et al., 1994).
4. There is an exponential cost of generating detectors with relation to the size of the self set (S). In order to overcome this drawback, D'haeseleer et al. (1996) proposed two novel detector-generating algorithms that run in linear time with respect to the size of the input. The authors also discussed how to set parameters for the algorithms, i.e., repertoire size, probabilities of match and failure in detection, practical issues, and rules of thumb for *r*-contiguous bits matching rule.

Evolutionary Algorithms

Evolutionary algorithms form a class of problem-solving methods inspired by the Darwinian theory on the origin of the species using natural selection (1859). The evolutionary approach simulates variation, competition, and selection between and among solutions in a population.

Basic Evolutionary Computation

The selection of the representation scheme or encoding of the possible solutions (e.g., binary or real value), as well as the selection of the evaluation function and variation operators are essential procedures used to implement an evolutionary algorithm. The following steps must be executed:

1. An initial population of possible solutions is required as a start point of the artificial evolution.

2. Each individual is evaluated by using a fitness (evaluation) function that is specific to the problem being solved. The fitness function leads to favoring more qualified individuals or parents to be used for reproduction.
3. The parents are recombined and/or mutated in order to obtain offspring.

The steps (2) and (3) are repeated until an explicit condition is satisfied. When the program halts, only the more qualified solutions will have survived.

Evolutionary Algorithms Approaches

There are four commonly known approaches of evolutionary algorithms: Genetic algorithms (Bremermann, 1962; Fraser, 1957; Holland, 1975), genetic programming (Koza, 1992, 1994), evolution strategies (Rechenberg, 1973), and evolutionary programming (Fogel et al., 1966). The type of operators selected, i.e., representation schemes, the reproduction operators, and the selection methods, establish the differences between each of these approaches, but over time they have hybridized and there is little scientific rationale for using any of these terms anymore.

Genetic Algorithms

Canonical genetic algorithms are a type of evolutionary algorithm in which individuals are represented by fixed-length bit strings. Each position in the string represents a specific feature of an individual. By applying the biological analogy, each bit would be equivalent to a gene from a chromosome. Parents are selected for crossover probabilistically, based their fitness values. Bit-string crossover and bit-flipping mutation are used typically as variation operators. Bit-string crossover combines partial characteristics of two strings to create a new string. Bit-flipping mutation consists of a

single bit in the string being flipped. The offspring created by crossover and mutation generally replace the parents, depending on the fitness criteria and the method of selection. Figure 4 illustrates bit-string crossover of parents (a) and (b) to form offspring (c) and (d). Figure 5 illustrates bit-flipping mutation.

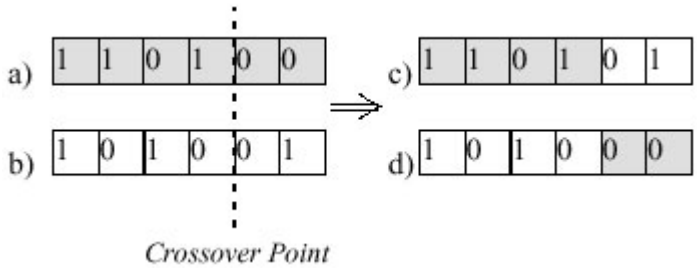


Figure 4. Bit-string crossover

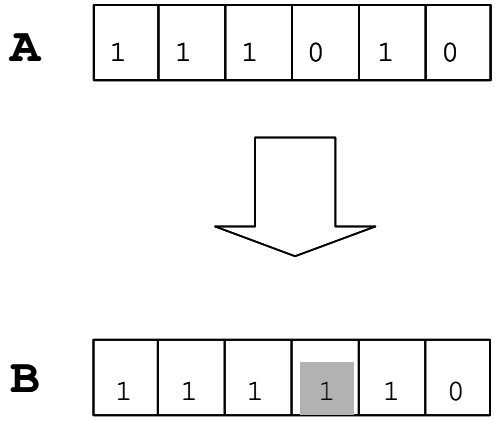


Figure 5. Bit-flipping Mutation of Parent A to form B (Offspring)

Genetic Programming

Genetic programming is another type of evolutionary algorithm and a subset of genetic algorithms, where the representation used is a variable-sized tree of functions and

values. The data structures that go through variation are representations of computer programs. The execution of the programs represents the evaluation function. In a genetic programming representation, each leaf in the tree constitutes a value from an available set of values. An entire tree corresponds to a single function that may be evaluated. Usually, the evaluation of the tree is in a leftmost, depth-first manner.

Except in the characteristics, i.e., representation scheme and variation operators, mentioned previously, genetic programming algorithms usually are similar to the other approaches, i.e., genetic algorithm, evolution strategies, and evolutionary programming.

Figure 6 illustrates a tree structure representing the expression (+ x (/ 5 y)) in genetic programming.

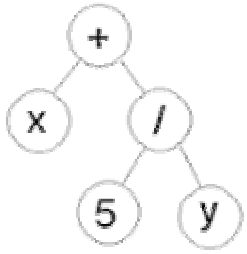


Figure 6. Genetic programming representation

Evolution Strategies

The fixed-length real-valued vector is the representation used by evolution strategies. Similar to the bit strings of genetic algorithms, each representation in the vector corresponds to a particular feature of the individual. Parents are selected at random. Intermediate recombination and Gaussian mutation are the operators used typically for variation. Intermediate crossover consists of having the values of each

position of the vector average together to form a new offspring. The Gaussian mutation is the adding of a Gaussian distribution to each element to form a vector (individual).

Evolutionary Programming

In this stochastic optimization strategy, a population of individuals is generated randomly. Each individual is mutated to create new individuals. Mutations are varied depending of the effect on the behavior of the individual. There is generally no mixing between the individuals of the population; primarily mutation operators are used (although recombination operators were proposed in Fogel et al, 1966). Probabilistic techniques, based upon fitness values, are used to select parents. Figure 7 shows a flowchart of the evolutionary programming strategy.

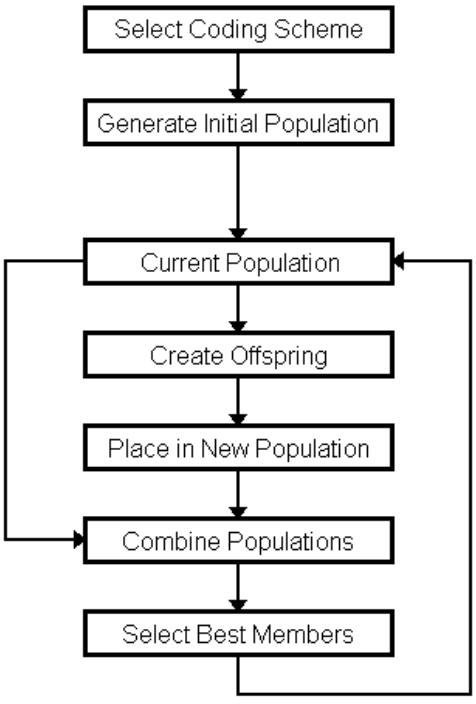


Figure 7. A flowchart of the evolutionary programming strategy

Self-adaptation

Self-adaptation is the term used to describe the process by which strategy parameters of evolutionary algorithms are encoded into the individual and adjusted by empirical rules of variation. For example, if an individual is represented as ($x_{1,}..., x_n$, $\sigma_1,\ldots, \sigma_n$) then the self-adaptation of the mutation step size of the detector can be completed by using the following formulas:

$$\sigma_i^{'} = \sigma_i \times e^{N(0,\tau_0)}$$

$$x_i^{'} = x_i + N(0,\sigma_i^{'})$$

Where i is the index identifying the component of x, and τ_0 is a constant.

Adaptive evolutionary computations are classified into three basic classes: Population-level, individual-level, and component-level, depending on the association that exists between the adaptive parameters and the evolutionary process (Angeline, 1995, p.155).

Population-level adaptations adjust strategy parameters that are global to the entire population, for example, the frequency of crossover. Individual-level adaptive methods associate strategy parameters with each individual to determine how the algorithm manipulates the individual, e.g., by mutation, crossover, and/or selection. Component-level adaptation also associates strategy parameters with each population member and additionally affects how each representational component of the individual is modified. "At all levels the goal is the same: to dynamically adjust the values of the strategy parameters to bias the production of offspring toward more appropriate solutions more quickly" (Angeline, 1995, p.155). Typically, strategy parameters are self-adapted to the level of individuals.

Approaches for Self-Adaptation of Strategy Parameters

Mutation Operators

The technique of self-adaptation has been used to deal with parameters related to the mutation operator, specifically for the variances and co-variances of a generalized *n*-dimensional normal distribution, as introduced by Schwefel (1977) and Fogel (1992) for the purpose of parameter optimization variants of evolution strategies and evolutionary programming. However, the technique has been used successfully in other search spaces also, such as binary search spaces, integer search spaces, and finite state machines (Bäck, Fogel, & Michalewicz, 1997).

Crossover Operators

The application of self-adaptation to recombination in evolutionary algorithms has been frequent used than with mutation operators, a circumstance that "can be explained in part by the historical emergence of the algorithmic principle in connection with the mutation operator in evolution strategies and evolutionary programming" (Bäck, Fogel, & Michalewicz, 1997, p. C7.1:12). This described approach has dealt mainly with either application probability or segment exchange probability.

Convergence Analysis of Algorithms of the Form $(\mu + \lambda) - EA$

As SANSAD is an instance of a $(\mu + \lambda) - EA$, which is demonstrated in Chapter 4, the analysis of its convergence properties is illustrated with the work of Bäck, Rudolph, & Schwefel (1993) on the convergence properties of the algorithms of that type.

Global Convergence Analysis

In a simple way, Bäck, Rudolph, & Schwefel (1993) proved the global convergence of algorithms of the form $((\mu+\lambda)-EA)$ by using the Borel-Cantelli Lemma as follows.

Definition 1

An optimization problem of the form

$$f^* := f(x^*) := \min\{ f(x) \mid x \in M \subseteq R^n \} \qquad (1)$$

is called a regular global optimization problem if and only if

$$\begin{array}{ll} (A1) & f^* > -\infty, \\ (A2) & x^* \in \operatorname{int}(M) \neq 0 \text{ and} \\ (A3) & \mu\left(L_{f^*+\epsilon}\right) > 0 \end{array}$$

Here, f is called the *objective function*, M is the *feasible region*, f^* the *global minimum*, x^* is the *global minimizer* or *solution*, μ is the number of parents, and $L_a := \{x \in M \mid f(x) < a\}$ the *lower level set* of f.

For later use, algorithms of type $(\mu+\lambda)-EA)$ are represented as follows:

$$\mathrm{X}_{t+1} = \mathrm{Y}_{t+1} \cdot 1_{L_{f(x_t)}}(\mathrm{Y}_{t+1}) + x_t \cdot 1_{L^c_{f(x_t)}}(\mathrm{Y}_{t+1}) \qquad (2)$$

where the indicator function $1_A(x)$ is one if $x \in A$ and zero otherwise and where Y_{t+1} is a random vector with some probability density. Generally p_Z is chosen as a *spherical* or *elliptical distribution*:

Definition 2

A random vector *z* of dimension *n* is said to possess an *elliptical distribution* if and only if it has stochastic representation $z \overset{d}{=} rQ^{i}u$, where the random vector *u* is uniformly distributed on a hypersphere surface of dimension *n* stochastically independent to a nonnegative random variable *r*, and where matrix $Q : k \times n$ with rank $\left(Q^{'}Q\right) = k$. If $Q : n \times n$ and $Q = I$, then *z* is said to possess a *spherical (symmetric) distribution*.

The above definition shows that an EA-type algorithm using a spherical or elliptical variation distribution is equivalent to a random direction method with some step size distribution.

Global Convergence Proof

THEOREM 1 (Baba, 1981; Pintér, 1984; Solis & Wets, 1981)

Let $p_t := P\left\{ \; X_t \in L_{f^* + \in} \; \right\}$ be the probability to hit the level set $L_{f^* + \in}, \in > 0$ at step *t*. If

$$\sum_{t=1}^{\infty} p_t = \infty \qquad (3)$$

then $f\left(X_t\right) - f^* \to 0$ a.s. for $t \to \infty$ or equivalently

$$P\left\{ \; \lim_{t \to \infty} \left(f\left(X_t\right) - f^* \right) = 0 \; \right\} = 1$$

for any starting point $x_0 \in M$.

LEMMA 1

Let C be the support of stationary p_Z. Then it holds that: $M \subseteq C$ and M bounded $\Rightarrow L_a$ bounded $\forall a \leq f(x_0) \Rightarrow p_t \geq p_{\min} > 0$ for all $t > 0 \Rightarrow \liminf_{t\to\infty} p_t > 0 \Rightarrow (3)$

If condition (3) is not fulfilled then it can be concluded that the probability to convergence on the global minimum for any starting point $x_0 \in M$ with increasing t is zero as pointed out by Pintér (1984).

Convergence Rates Analysis

Bäck, Rudolph, and Schwefel (1993) analyze the convergence rates of algorithm of type (2) as follows.

Definition 3

The value $\delta_t := \mathrm{E}\left[f(X_t - f^*) \right]$ is said to be the expected error at step t. An algorithm has a *sublinear convergence rate* if and only if $\delta = \mathrm{O}(t^{-b})$ with $b \in (0,1]$ and a *geometrical convergence rate*, if and only if $\delta = \mathrm{O}(t^r)$ with $r \in (0,1)$.

THEOREM 2 (Rappl, 1984; Rappl, 1989)

Let f be a (l,Q) - strongly convex function, i.e., f is continuously differentiable and with $l > 0, Q \geq 1$ there holds for all $x, y \in M$

$$l \|x-y\|^2 \leq (\nabla f(x) - \nabla f(y))'(x-y) \leq Ql \|x-y\|^2$$

and $Z \overset{d}{=} RU$, where R has nonvoid support $(0,a) \subseteq R$. Then the expected error of an algorithm (2) decreases for any starting point $x_0 \in M$ with the following rates:

$$\mathrm{E}\left[\quad f(X_t) - f^* \quad\right] \leq \begin{cases} O\left(t^{-2/n}\right) & , p_Z \text{ stationary} \\ O\left(\beta^{-t}\right) & , p_Z \text{ adapted} \end{cases}$$

with $\beta \in (0,1)$

The adaptation of the steps sizes is to be made as follows: $R_{t+1} = \|\nabla f(x_t)\| R$. Rappl (1984, pp. 102-143) shows that the step size can be adapted via a success/failure. If there is a failure, the step size is decreased with a factor $\gamma_1 \in (0,1)$; otherwise, if there is a success, a factor $\gamma_2 > 1$ increases the step size. After that, convergence follows for $\gamma_1 \gamma_2 > 1$.

The expected number of steps to achieve certain accuracy may differ significantly with geometrical convergence, e.g., the number of steps needed if $\beta = 0.9$ or if $\beta = 1 - 10^{-10}$, which implies that the search for optimal step size schedules should not be neglected.

EXAMPLE 1

Let $f(x) = \|x\|^2$ with $M = R^n$ be the problem. Evidently, f is $(2,1)$ – convex:

$$\left(\nabla f(x) - \nabla f(y)\right)'(x - y) = 2\|x - y\|^2$$

The mutation vector *z* in algorithm (5) is chosen to be multinormally distributed with zero mean and covariance matrix $C = \sigma^2 I$. Therefore, the distribution of the objective function values is expressed as $f(x_t + Z_t) \sim \sigma^2 X_n^2(k)$, where $X_n^2(k)$ denotes a noncentral X^2 - distribution with *n* degrees of freedom and noncentrality parameter $k = \dfrac{\|x_t\|^2}{\sigma^2}$.

Using the fact that (Johnson & Kotz, 1970, p. 135)

$$\frac{x_n^2(k)-(n+k)}{\sqrt{2(n+2k)}} \to N \sim N(0,1)$$

for $n \to \infty$, the limiting distribution of the normalized variation of the normalized variation of objective function values V becomes

$$\begin{aligned} V &:= \frac{f(x_t)-f(X_{t+1})}{f(x_t)} \\ &= 1-\frac{\sigma^2}{\|x_t\|^2} X_n^2(k) \\ &\to 1-\frac{\sigma^2}{\|x_t\|^2}\left(n+k+\sqrt{2n+4k}N\right) \\ &= -\frac{s^2}{n}-\frac{s^2}{n}\sqrt{\frac{2}{n}+\frac{4}{s^2}}N \\ &= -\frac{s^2}{n}-\frac{2s}{n}N \end{aligned}$$

with $\sigma = \frac{s\|x_t\|}{n}$. Since algorithm (2) accepts only improvements, the analysis is focused on the expectation of the random variable $V^+ = \max\{0,V\}$:

$$\begin{aligned} E\left[V^+\right] &= \int_0^\infty \frac{nu}{2s\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{nu+s^2}{2s}\right)^2\right] du \\ &= \frac{1}{n}\left\{ s\sqrt{\frac{2}{\pi}}\exp\left(-\frac{s^2}{8}\right) - s^2\left[1-\Phi\left(\frac{s}{2}\right)\right]\right\} \end{aligned}$$

where $\Phi(\cdot)$ denotes the cumulative density function of a unit normal random variable. The expectation becomes maximal for $s^* = 1.224$ such that $E\left[V^+\right] = \frac{0.404}{n}$ and

$$\sigma^* = \frac{1.224}{n}\|x\| \qquad (4)$$

$$= \frac{1.224}{n}\sqrt{f(x)} \qquad (5)$$

$$= \frac{0.612}{n} \left\|\nabla f(x)\right\| \quad (6)$$

where (4) is the value also given by Rechenberg (1973) which is converted to (5) and (6) in the notation of Fogel (1992) and Rappl (1984), respectively. Evidently, if f is modified to $f_a(x) = \|x\|^2 + 1$, then control (5) will fail to provide geometrical convergence. Likewise, optimizing $f_b(x) = \|x - 1\|^2$ control (4) will fail whereas control (6) will succeed in all cases due to its invariance with respect to the location of the unknown global minimizer and the value of the unknown global minimum. Additionally, the dependence on the problem dimension n is of importance: Omitting this factor, geometrical convergence can still be guaranteed, but it will be very slow compared to the optimal setting.

Contribution to the Field

The novel self-adaptive evolutionary negative selection approach will be a useful alternative to the identification of network activity patterns that exhibit a behavior different from that of the normal user (anomaly detection). Furthermore, the novel self-adaptive evolutionary negative selection approach will provide a solution to the time/size limitation of the process to generate the set of candidate detectors required by the negative approach, a problem that has motivated the creation of several algorithms, e.g., linear, greedy, binary template, and the NSMutation.

Summary of the Chapter

This chapter presented the immune system as a network of different kinds of cells, B-lymphocytes and T-lymphocytes, which participate in the actual biological immune response. The B-lymphocytes are produced by the bone marrow, whereas T-lymphocytes are produced by the thymus. The mechanisms for the generation of B-lymphocytes and T-lymphocytes has inspired the development of algorithms, i.e., negative and positive selection, that can be used in different domains where the differentiation between self and non-self is required, e.g., pattern recognition and the security of information systems. Likewise, clonal selection and immune network theories inspired the development of the evolutionary characteristics of other artificial immune systems algorithms that are used to solve different real-life problems.

Evolutionary algorithms were described as a class of problem-solving methods inspired by the Darwinian theory on the origin of the species via natural selection. Evolutionary algorithms are in contrast to deterministic problem solving methods that process only a single solution at one time. The four most common evolutionary algorithms are genetic algorithms, genetic programming, evolution strategies, and evolutionary programming, and they were explained briefly. The chapter concluded by defining the self-adaptation approach as a mechanism that could alter dynamically the strategy parameters that control evolutionary algorithms, describing mutation and crossover operators as approaches for the self-adaptation of strategy parameters, and providing an analysis of the convergence properties of algorithms of the form $(\mu + \lambda) - EA$.

Chapter 3

Methodology

Research Methodology

The experimental-quantitative method was the main approach used for the research presented in this dissertation. This approach facilitated the analysis and explanation of the results obtained from manipulating variables, e.g., matching threshold. This methodological strategy was chosen after considering the intrinsic measurable nature of the topic (time of generation versus quality of the detectors) and the research questions/objectives and an examination of the needs of the hypothesis presented in this dissertation.

Specific Procedures

Procedure 1: Development, Design, and Implementation of the Algorithm

This procedure included the development, design, and implementation of the algorithm from a conceptual and technical perspective. On the conceptual side, evolutionary and adaptive concepts were used to create a self-adaptive negative selection algorithm that can work efficiently when large data sets are required (e.g., anomaly detection in network environments). One of the key approaches for the work presented in this dissertation was the use of adaptive mechanisms. By using adaptation, domain-specific information and multiple variation operators could be incorporated into the evolutionary algorithm more easily. Furthermore, the algorithm itself was allowed to select those values and operators that provided the best results.

On the technical side, the codification of the algorithm and other components presented in this dissertation was performed in Java 2 Platform Standard Edition – version 1.4.2 01, and Oracle RDBMS version 9.2.0.1.0 on Windows 2000. Data migration from the comma delimited format (IES datasets) to the Oracle database was completed using SQL Loader version 9.2.0.1.0.

Initial Population

The initial population of integer detectors was generated using the method Random.nextInt (int n) provided by Java. This method returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive). For the experiments presented in this paper, $specifiedValue = stringLength - 1$, e.g., the specified value for a binary string of 8 bits would be $256 - 1 = 255$.

Representation (Encoding)

Binary and real values were the two forms of encoding used to represent the individuals (chromosomes) evaluated and processed by the self-adaptive negative selection algorithm. In binary encoding, every chromosome is a string of bits: 0 or 1, e.g., the decimal number 157 would be represented as the binary number 10011101. In this case, the whole string represents one and only one individual (chromosome). In real-valued encoding, no conversion is required, e.g., the decimal number 157 would be represented as 157.

Evaluation Function

The experiments presented here used the *r*-contiguous bits matching rule (the number of continuous bits that differ between two binary strings), which captured the matching between lymphocyte receptors and antigens in the immune system (Percus, Percus, & Perelson, 1993), and the Hamming distance matching rule (the number of bits that differ between two binary strings). The affinity of each detector was calculated individually against the entire self-population. Once the individual cycle was completed, then the affinity mean was calculated and assigned as the affinity of the detector. The arithmetic mean was a suitable representation that encompassed all the affinity evaluations made.

Parent Selection (Reproduction)

The algorithm used tournament selection to select the best parents.

- Two (2) individuals were selected randomly to form a group (binary tournament selection).
- The individual with the highest fitness from the group was selected.
- The process was repeated to select the second parent. A small tournament size of two was chosen because it was the simplest to implement. Other tournament sizes were not tried.

Recombination or Crossover (Reproduction)

Single-point crossover was used; one crossover point was selected, the binary string from the beginning of the chromosome to the crossover point was copied from the first parent, and the rest was copied from the other parent.

The method Random.nextInt (int n) provided by Java 2 Platform Standard Edition – version 1.4.2.01 – determined the single point, thus making it possible for the single point to change dynamically during the run of the algorithm. Figure 8 illustrates random crossover.

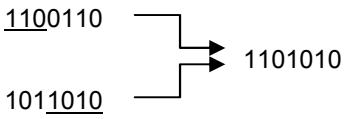


Figure 8. Example of random crossover for a single-point = 3

Mutation – Self-adaptation of the Mutation Step Size (Reproduction)

Mutation was realized according to equations presented in Table 1. Each individual was represented as ($x_1,\ldots, x_n, \sigma_1,\ldots, \sigma_n$).

Equation #	Equation	Comments
7	$\sigma_i = \sigma_i \times e^{\tau \times N(0,1)}$	where τ is a constant determined by equation (9)
8	$x_i = x_i + N(0, \sigma_i)$	
9	$\tau = (2n)^{-\frac{1}{2}}$	where *n* is the dimension of the problem, i.e., the dimension would be eight for a binary string of eight bits.

Table 1. Equations for the self-adaptation of the mutation step size

Gaussian mutation was used to mutate the individual (chromosome) and the mutation step size in the self-adaptive negative selection algorithm as follows:

1. The binary offspring is converted to an integer.
2. The mutation step size (standard deviation) of the integer value obtained in (1) is determined.

3. The mutation step size obtained in (2) is used. The mutation is completed as follows:
   - Calculate $\tau = (2n)^{-\frac{1}{2}}$, where *n* is the number of dimensions of the problem, e.g., the dimension for a binary string of eight bits is eight.
   - Calculate Gaussian noise to mutate step size ($e^{\tau \times N(0,1)}$). Gaussian noise is obtained from a standard normally distributed random variable, scaled by τ, and then taken as e to that power.
   - Mutate step size by multiplying step size by the determined quantity.
4. The chromosome is mutated by adding the mutation step size calculated in the previous step to the integer value of the offspring obtained in (1). If the new integer chromosome is greater than the MAXINTEGER (e.g., 255 is the maximum integer number that can be represented with a binary string of 8 bits), then the chromosome is truncated from right to left to a binary string of 8 bits.
5. The affinity (fitness) of the offspring is calculated against the entire population of self. If the affinity of the offspring is better than the affinity of the best parent, then the offspring replaces the best parent.

Procedure 2: Proof that SANSAD is an instance of a $(\mu + \lambda) - EA$ *with elitist selection*

This procedure implied a proof that SANSAD is an instance of a $(\mu + \lambda) - EA$ with elitist selection, and that consequently it has asymptotic global convergence properties and certain convergence rate properties on special functions.

Procedure 3: Data Collection

Three datasets provided by the "Information Exploration Shootout" were used for experimentation. The content of each dataset is detailed as follows:

- Dataset 1 contained normal network activity (IP addresses) with no intrusions and represented the self data.
- Datasets 2 and 3 contained normal activity and some intrusions. The intrusions represented the non-self data. Each dataset was of a different size, which facilitated evaluating the behavior of the algorithms. Using the equations provided by Forrest et al. (1994), the algorithm calculated the values for the repertoire size (N_{R0}), the probability of matching (P_m), and the probability of failure (P_f).

In total, 12 scenarios were created to test the novel self-adaptive evolutionary negative selection algorithm for anomaly detection and NSMutation algorithms. Table 2 shows the variants of each scenario.

Scenario	Dataset	Algorithm	Matching Rule	Variation
1	2	SANSAD	*r*-contiguous bits	Mutation
2	2	SANSAD	*r*-contiguous bits	Recombination and Mutation
3	2	SANSAD	Hamming distance	Mutation
4	2	SANSAD	Hamming distance	Recombination and Mutation
5	3	SANSAD	*r*-contiguous bits	Mutation
6	3	SANSAD	*r*-contiguous bits	Recombination and Mutation
7	3	SANSAD	Hamming distance	Mutation
8	3	SANSAD	Hamming distance	Recombination and Mutation
9	2	NSM	*r*-contiguous bits	N/A
10	2	NSM	*r*-contiguous bits	N/A
11	3	NSM	Hamming distance	N/A
12	3	NSM	Hamming distance	N/A

Table 2. Variants of each scenario used for testing the self-adaptive evolutionary negative selection (SANSAD) and NSMutation (NSM) algorithms

The NSMutation generated a file containing statistical results of the monitoring phase. A database developed in Oracle version 9.2.0.1.0 was used to store the datasets

for the experimentation and the activity generated by the self-adaptive evolutionary negative selection algorithm, including: scenarios (parameters used), trials (trial duration, total detections by trial, total false detection, and total true detection), and non-self data detected. Figure 9 shows the entity relation model of the self-adaptive evolutionary negative selection algorithm. The Entity-Relation model was included to illustrate the type of information to be stored in the database.

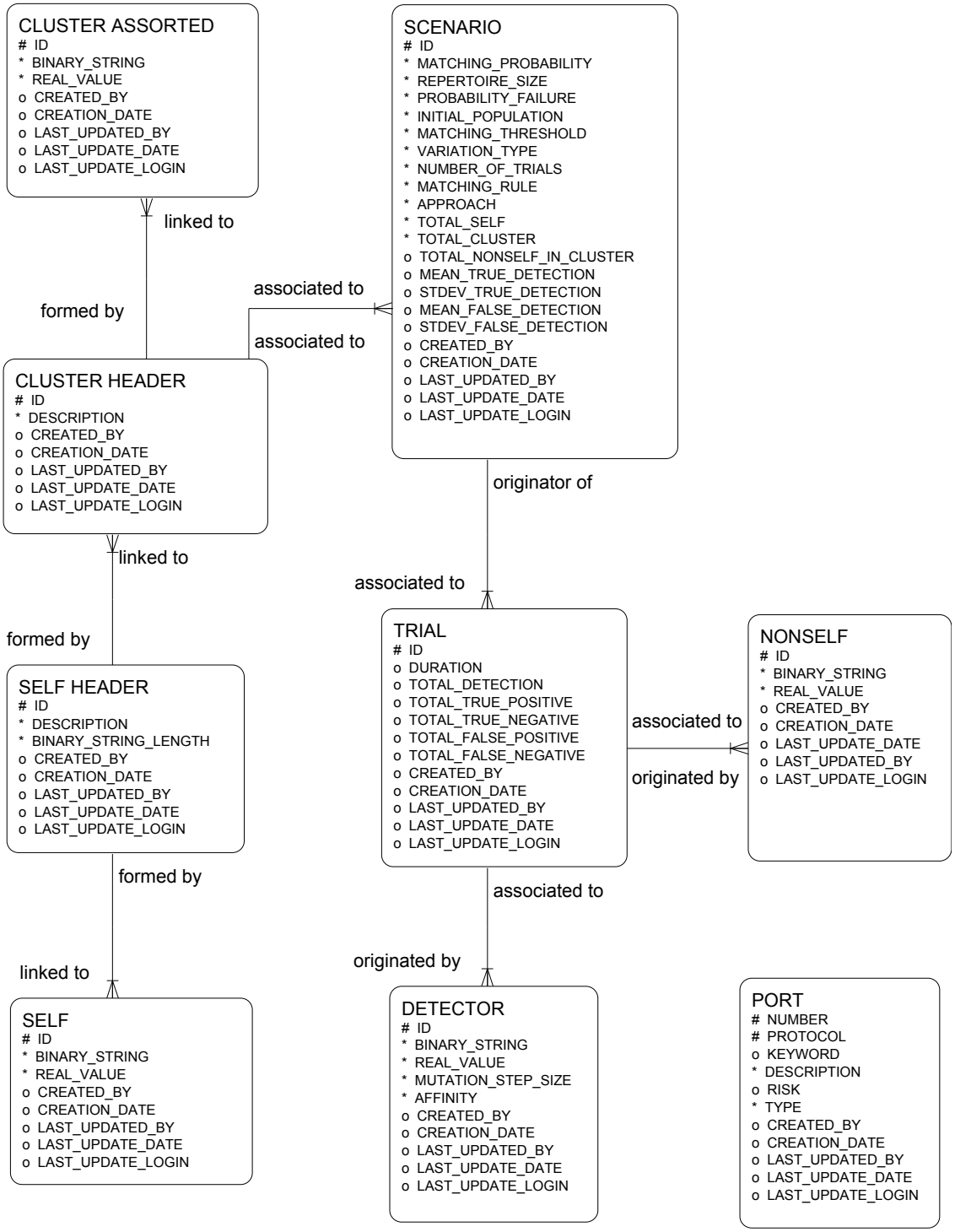


Figure 9. The entity relation model of the self-adaptive evolutionary negative selection algorithm

Selection and Profiling of Datasets

The Information Exploration Shootout project provided five datasets in comma-delimited format, which for identification purposes were called base, net1, net2, net3, and net4. The first dataset (base) contained normal network activity (IP Addresses) with no intrusions that represented the self data. The other four datasets, i.e., net1, net2, net3, and net4, contained normal activity and some intrusions that represented the non-self data. The format of each line of the datasets was represented as follows: time, src_addr, src_port, dest_addr, dest_port, flag, seq1, seq2, ack, win, buf, ulen, op. Table 3 describes each one of the fields of the structure used by the datasets of the Information Exploration Shootout Project.

Field	Description
Time	Converted to floating point seconds (hr × 3600+min × 60+secs)
Addr and port	The first two fields of the src and dest address make up the fake address, so the converted address was made as x + y × 256.
Flag	Added a "U" for udp data (only has ulen) X –means packet was DNS name server request or response. The ID# and rest of the data is in the "op" field. XPE –meaning there were no ports from "fragmented packets"
seq1	The data sequence number of the packet
seq2	The data sequence number of the data expected in return
Buf	The number of bytes of received buffer space available
Ack	The sequence number of the next data expected from the other direction on the connection
Win	The number of bytes of received buffer space available from the other direction on the connection
Ulen	The length, if a udp packet
Op	Optional info, such as (df) … do not fragment

Table 3. Structure description of the datasets of the Information Exploration Shootout Project

The migration, selection, and profiling of the datasets were completed as follows:

1. The datasets were migrated from comma-delimited format into an Oracle interface table using SQL Loader.
2. Five SQL views were created to select a unique combination of the src_addr + dest_addr + dest_port (network activity), which formed a real number that could be represented with a binary string of 32 bits. All the records were selected from the datasets except those records whose src_port = 80, which were outgoing Internet connections (HTTP protocol).
3. The tables SELF_HEADER, SELF, CLUSTER_HEADER, and CLUSTER_ASSORTED that belonged to SANSAD's database were populated using as a source the views created in (2).
4. Net1 and Net2 were chosen after the ratio of self records and non-self records for each one of the datasets was calculated and analyzed. Net1 was selected because it was the largest dataset of the group with the highest proportion (11/224 or 4.91%), and Net2 because it possessed the highest proprotion, i.e., 7/52 (13.46%) when compared to the other datasets. Table 4 shows the proportions for each one of the datasets evaluated.

Dataset	Proportion/Percentage
Net1	11/224 = 4.91%
Net2	7/52 = 13.46%
Net3	7/214 = 3.27%
Net4	This dataset does not contain self records.

Table 4. Proportion of self vs. non-self records

Setting matching threshold (r)

The matching threshold optimal value was obtained by changing values of r from 1 to l (length of string). The value selected for the r-contiguous bits and Hamming Distance matching rules was 8 and 19, respectively. Those values were selected because they had the lowest proportion of false detections versus true detections (mean false detections / mean true detections). The steps used to set the matching threshold in this work can be summarized as follows:

1. The datasets containing binary strings were selected with a fixed length of 32 bits: (1) self records; (2) assorted records (self and non-self).
2. The algorithm for each matching rule (r-contiguous bits and Hamming distance) was executed 25 times, changing values of r from 1 to 32 (binary string length) and using a theoretical setting for the repertoire size (N_{R0}), probability of match (P_m), and probability of failure (P_f).
3. The true and false detections were evaluated for each result obtained in (2).
4. The value of r that maximizes the true detections and minimizes the false detections was selected.

Table 5 shows the results obtained for different values of r using Net1 and the r-contiguous bits matching rule, where TD = mean true detections, and FD= mean false detections. The algorithm could not calculate the values of the parameters when $1 \leq r \leq 7$; hence, these values were not included in the table.

r	*TD*	*FD*	*FD/TD*
8	211.56	11.04	0.052
9	195.64	11	0.056
10	195.88	13.4	0.068
11	185.68	14.24	0.077
12	149.88	14.4	0.096
13	114.16	13.2	0.116
14	134.36	14.72	0.11
15	45.2	10.12	0.224
16	68.36	11.68	0.171
17	91.91	12.92	0.141
18	67.72	11.56	0.171
19	75.68	12.04	0.159
20	43.32	10.28	0.237
21	59.48	11.16	0.188
22	75.64	12.04	0.159
23	83.72	12.48	0.149
24	75.64	12.04	0.159
25	59.48	11.16	0.188
26	75.64	12.04	0.159
27	75.64	12.04	0.159
28	43.32	10.28	0.237
29	59.48	11.16	0.188
30	75.64	12.04	0.159
31	99.88	13.36	0.134
32	11	8.52	0.775

Table 5. Results of setting *r* when the *r*-contiguous bits matching rule was used

Table 6 shows the results obtained for each variant of *r* using Net1 and the Hamming Distance matching rule. Table 6 again only includes results for $8 \leq r \leq 32$ since the algorithm could not calculate theoretically the values of the parameters, for $1 \leq r \leq 7$.

R	*TD*	*FD*	*FD/TD*
9	213	11	0.052
10	213	11	0.052
11	213	11	0.052
12	213	11	0.052
13	212.88	11	0.052
14	212.88	11	0.052
15	204.56	11	0.054
16	211.04	11	0.052
17	202.16	10.76	0.053
18	167.8	8.84	0.053
19	143.56	7.08	0.049
20	111	6.48	0.058
21	69.76	3.88	0.056
22	56.32	4.56	0.081
23	27.28	6.32	0.232
24	28.8	8.48	0.294
25	11.36	8.28	0.729
26	27.32	9.32	0.341
27	11	8.52	0.775
28	35.24	9.84	0.279
29	11	8.52	0.775
30	19.08	8.96	0.47
31	27.16	0.4	0.346
32	19.08	8.96	0.47

Table 6. Results of setting r when the Hamming Distance matching rule was used

Testing the SANSAD algorithm

SANSAD was tested with datasets containing binary strings of 32 bits. Each scenario comprised 100 trials. The repertoire size (N_{R0}), probability of match (P_m), and probability of failure (P_f) were set theoretically. Table 7 shows the different scenarios used for testing SANSAD.

Dataset	Matching Rule	Variation	r	N_{R0}	P_m	P_f
Net1	*r*-contiguous bits	Mutation	8	83	0.0508	0.0147
Net1	*r*-contiguous bits	Recombination & Mutation	8	83	0.0508	0.0147
Net1	Hamming distance	Mutation	19	81	0	0.9988
Net1	Hamming distance	Recombination & Mutation	19	81	0	0.9988
Net2	*r*-contiguous bits	Mutation	8	83	0.0508	0.0147
Net2	*r*-contiguous bits	Recombination & Mutation	8	83	0.0508	0.0147
Net2	Hamming distance	Mutation	19	81	0	0.9988
Net2	Hamming distance	Recombination & Mutation	19	81	0	0.9988

Table 7. Variants of each scenario used for testing SANSAD

Testing the NSMutation algorithm

NSMutation was tested with the same datasets used for SANSAD. The repertoire size (N_{R0}), probability of match (P_m), and probability of failure (P_f) were set manually, using the values calculated by SANSAD in previous experiments. The mutation probability (mutProb) and detector life-time indicator (mutLim) were set according to the values used by Ayara et al. (2002a). Table 8 shows the scenarios and parameter settings used for testing the NSMutation algorithm.

Dataset	Matching Rule	r	N_{R0}	P_m	P_f	*mutProb*	*mutLim*
Net1	*r*-contiguous bits	8	83	0.0508	0.0147	0.5	4
Net1	Hamming distance	19	81	0	0.9988	0.5	4
Net2	*r*-contiguous bits	8	83	0.0508	0.0147	0.5	4
Net2	Hamming distance	19	81	0	0.9988	0.5	4

Table 8. Variants of each scenario used for testing the NSMutation algorithm

Procedure 4: Data Description

Descriptive statistics were used to analyze the observed computation time, as well as true and false detections for each set of detectors generated by trial (data collected from the experiments) of the self-adaptive evolutionary negative selection and the NSMutation algorithms. The mean and standard deviation of the data collected were

computed. Self-explanatory tables containing descriptive statistics are provided in Chapter 4.

Procedure 5: Data Analysis

The main objective of this research was to test the hypothesis that a novel evolutionary negative selection algorithm for anomaly detection (in non-stationary environments) can generate detectors of better quality and with better computational time performance than the NSMutation algorithm by using self-adaptive techniques (dynamic parameter setting) to mutate the mutation step size of the detectors. Using the hypothesis tests (inferential statistics) included in the Analysis ToolPak of Microsoft Excel 2002 and following the steps listed below, several scenarios were compared to investigate the main hypothesis. Chapter 4 shows in detail the data analysis realized for each pair of scenarios, which followed as.

1. The *P*-value generated by the one-tailed test for the CT, TD, and FD of each set of SANSAD scenarios was evaluated. If the *P*-value was less than 0.05, then the null hypothesis was rejected in favor of the alternative. Otherwise, the null hypothesis was not rejected, because of insufficient evidence.
2. The best scenario for further comparison against the set of NSMutation scenarios was selected based on the analysis conducted in (1).
3. The *P*-value generated by the one-tailed hypothesis test for SANSAD and NSMutation scenarios was evaluated following the same rules explained in (1).
4. The main hypothesis was rejected or failed to be rejected based on (3).

Figure 10 illustrates how the different scenarios of SANSAD and NSMutation were compared. Two scenarios are statistically compared to form a group. The best

statistically significantly scenario from the group is selected for further comparison. The comparison is realized several times until only one scenario is obtained.

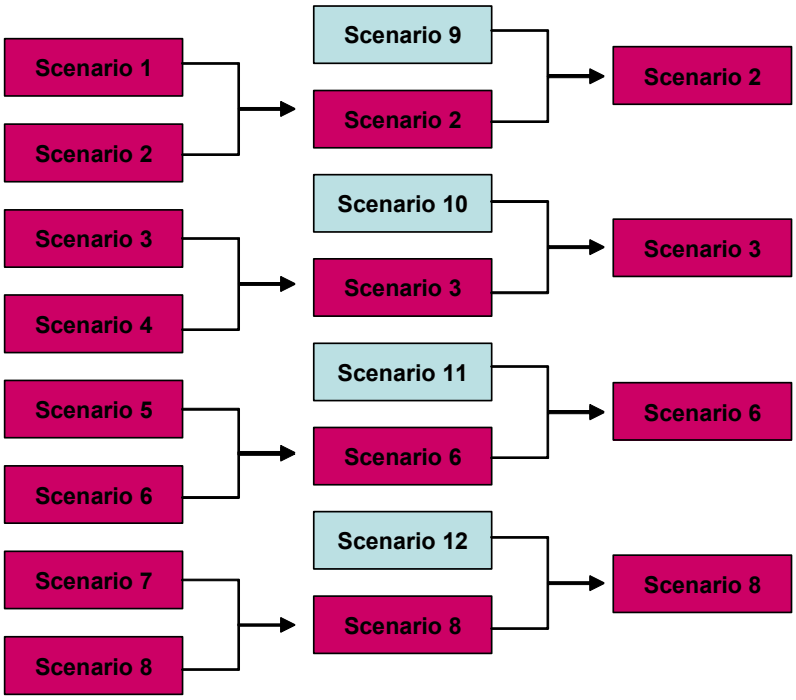


Figure 10. Chart of comparison of SANSAD and NSMutation scenarios.

Procedure 6: Conclusions

The conclusions of the study were stated based on the analyses performed and results achieved. Alternative explanations for the findings were discussed. Strengths, weaknesses, and limitations of the study also were delineated. Chapter 5 presents a comprehensive discussion of the conclusions of the research.

Formats for Presenting Results

Tables and bar graphs were used to present results. The tables were used to summarize the data, show its distribution, and indicate how the data were spread out over a range of values. In addition, the tables were helpful to visualize and comprehend

clusterings and trends in the data. Elsewhere, the bar graphs illustrated the computation time required to generate detectors versus the quality of the detectors of the novel self-adaptive evoluationary negative selection algorithm and the NSMutation algorithm. In this case, the mean percentage of true and false detections allowed for quantifiying the quality.

Projected Outcomes

The research presented in this dissertation was expected to demonstrate that a novel evolutionary negative selection algorithm for anomaly detection (non-stationary environments) could generate detectors of better quality and with better computational time performance than the NSMutation algorithm by using self-adaptive techniques (dynamic parameter settings) to mutate the mutation step size of the detectors.

Likewise, if the hypothesis described previously were true, then the following statements would be true:

1. The percentage of false detections would be lower, compared to the percentage of false detections of the detectors generated by the NSMutation algorithm.
2. The process of manually calculating the mutation step size (mutation probabilities) would no longer be required (this process is time consuming and impractical). Therefore, the self-adaptive approach could replace manual calculation.

Resource Requirements

This research used different sources, including the online library of the University of Phoenix, the Distance Library Services (DLS) of Nova Southeastern University

Libraries, and CiteSeer. These three digital libraries are databases of scientific and technical journals, newspapers, magazines, books, patents, trade publications, and statistical data.

The algorithm presented in this work was codified in Java Standard Edition version 1.4.2, supported by a database implemented on Oracle version 9.2.0.1.0 and Windows 2000.

Summary of the Chapter

The procedures used to test the self-adaptive evolutionary negative selection algorithm developed along with the NSMutation algorithm were explained. Conceptual and technical details of the design and implementation of SANSAD were provided and discussed. A description of the datasets used for the experiment was provided, together with an explanation of how the data were migrated, selected, and profiled. The method of setting the matching threshold optimal value for *r*-contiguous bits and Hamming distance matching rules was explained. The computerized tools used for the hypothesis tests (one-tailed test) of the different scenarios evaluated were mentioned. The projected outcomes of the research were delineated.

Chapter 4

Results

Introduction

This chapter presents the main results of the investigation: A proof that SANSAD is an instance of a $(\mu + \lambda) - EA$ with elitist selection, together with the findings of testing SANSAD under different scenarios. This experimentation enabled: (1) estimate the performance of SANSAD under a projected set of operating conditions, and (2) maintain better control over experimental conditions.

Proof that SANSAD is an instance of a $(\mu + \lambda) - EA$ with elitist selection

Given: Let *P* denote a SANSAD algorithm. Let *R* denote a $(\mu + \lambda) - EA$ with elitist selection. Let *Q* denote an algorithm with the following properties:

1. Two or more parents, μ, are involved to produce λ offspring by means of recombination and mutation.
2. The recombination is discrete and mutation can reach any state in the search space.
3. Each parent has the same chance to be selected to create offspring.
4. Parents compete with offspring for survival into the next generation.
5. Elitist selection is applied.

It then follows that *P* is an instance of *Q*, which is in turn an instance of *R*, and by consequence, *P* is an instance of *R*.

Proof: Each possible configuration of a SANSAD algorithm, *P*, has each of the five properties of *Q*. SANSAD does not operate with less than two parents, uses discrete recombination and mutation that can create any possible solution (albeit with low probability), applies equal chances for selecting parents to generate offspring, competes parents with offspring, and uses elitist selection to maintain the best solution at each generation. Construct a set **P** that contains all possible instances of SANSAD algorithms, *P*. Construct a set **Q** that contains all possible instances of algorithms *Q*. It follows therefore that $\mathbf{P} \subset \mathbf{Q}$.

Similarly, each possible element in **Q** is an instance of a (μ+λ)-EA with elitist selection. Construct a set **R** that contains all (μ+λ)-EAs with elitist selection. It follows therefore that $\mathbf{Q} \subset \mathbf{R}$, and by the transitive property, $\mathbf{P} \subset \mathbf{R}$. Therefore, every possible instance of a SANSAD algorithm is an instance of a (μ+λ)-EA with elitist selection.

Since SANSAD is, an instance of a $(\mu + \lambda) - EA$, it has asymptotic global convergence properties and certain convergence rate properties on special functions.

The completion of the above mathematical proof allows for continuation in the next section that contains the description and analysis of the data used to evaluate the efficiency of SANSAD under different scenarios.

Data Description

Tables 9 - 16 illustrate numerically relevant aspects of the distribution of the data for the different evaluated scenarios. From this section on, for simplicity, the following abbreviations will be used, CT = computation time; TD = true detections; and FD = false detections.

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	186.86	195.76	211.72	210.86	11	10.85
Standard Error	135.2849	135.5290	0.2613	0.3585	0.0348	0.0626
Median	50	60	213	213	11	11
Mode	50	60	213	213	11	11
Standard Deviation	1352.8486	1355.2896	2.6134	3.5845	0.3482	0.6256
Sample Variance	1830199.3943	1836809.9822	6.8299	12.8489	0.1212	0.3914
Kurtosis	99.9981	99.9039	2.4911	1.2733	58.2655	11.9667
Skewness	9.9999	9.9929	-1.9043	-1.5646	5.8618	-0.8934
Range	13540	13560	10	12	4	6
Minimum	40	50	203	201	10	8
Maximum	13580	13610	213	213	14	14
Sum	18686	19576	21172	21086	1100	1085
Count	100	100	100	100	100	100

Table 9. Descriptive statistics for Scenario 1 (x) and 2 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	189.77	184.85	44.5	44.59	6.9	6.97
Standard Error	132.1451	129.0526	0.0927	0.1016	0.0389	0.0300
Median	50	50	45	45	7	7
Mode	50	50	45	45	7	7
Standard Deviation	1321.4508	1290.5262	0.9266	1.0160	0.3892	0.3000
Sample Variance	1746232.1587	1665457.9672	0.8586	1.0322	0.1515	0.0900
Kurtosis	99.9013	99.9013	3.7240	12.9307	32.1232	25.2722
Skewness	9.9927	9.9927	-2.0211	-3.3457	-5.1381	1.4524
Range	13229	12918	4	6	3	3
Minimum	40	40	41	39	4	6
Maximum	13269	12958	45	45	7	9
Sum	18977	18485	4450	4459	690	697
Count	100	100	100	100	100	100

Table 10. Descriptive statistics for Scenario 3 (x) and 4 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	82.08	76.99	184.71	105.2	10.02	5.82
Standard Error	2.8978	3.6851	1.0704	4.0127	0.0899	0.1872
Median	80	70	182	92	10	5
Mode	70	60	181	213	10	5
Standard Deviation	28.9785	36.8512	10.7038	40.1273	0.8987	1.8715
Sample Variance	839.7511	1358.0100	114.5716	1610.2020	0.8077	3.5026
Kurtosis	15.8990	19.0948	0.3153	1.4780	0.0556	1.5725
Skewness	3.1051	3.9944	0.7707	1.5754	-0.6361	1.4652
Range	241	260	48	153	4	7
Minimum	10	40	165	60	7	4
Maximum	251	300	213	213	11	11
Sum	8208	7699	18471	10520	1002	582
Count	100	100	100	100	100	100

Table 11. Descriptive statistics for Scenario 5 (x) and 6 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	87.33	78.61	35.32	25.66	5.49	3.54
Standard Error	3.6072	4.5790	0.3432	0.7231	0.1078	0.1617
Median	80	70	35	23	6	3
Mode	90	70	37	22	6	2
Standard Deviation	36.0723	45.7902	3.4315	7.2310	1.0777	1.6170
Sample Variance	1301.2132	2096.7454	11.7754	52.2873	1.1615	2.6145
Kurtosis	12.1165	10.0498	0.5517	0.0475	-0.5037	-0.2689
Skewness	2.9066	3.1921	0.6815	0.8597	-0.3442	0.9434
Range	250	260	16	30	4	5
Minimum	10	20	29	15	3	2
Maximum	260	280	45	45	7	7
Sum	8733	7861	3532	2566	549	354
Count	100	100	100	100	100	100

Table 12. Descriptive statistics for Scenario 7 (x) and 8 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	6956.51	195.76	33.28	210.86	190.72	10.85
Standard Error	81.2572	135.5290	0.3957	0.3585	0.3957	0.0626
Median	6885	60	33	213	191	11
Mode	7030	60	34	213	190	11
Standard Deviation	812.5725	1355.2896	3.9570	3.5845	3.9570	0.6256
Sample Variance	660274.0100	1836809.9822	15.6582	12.8489	15.6582	0.3914
Kurtosis	0.9849	99.9039	0.0515	1.2733	0.0515	11.9667
Skewness	0.5701	9.9929	0.2468	-1.5646	-0.2468	-0.8934
Range	4847	13560	19	12	19	6
Minimum	4897	50	24	201	181	8
Maximum	9744	13610	43	213	200	14
Sum	695651	19576	3328	21086	19072	1085
Count	100	100	100	100	100	100

Table 13. Descriptive statistics for Scenario 9 (x) and 2 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	6886.08	184.85	13.04	44.59	38.96	6.97
Standard Error	76.07927	129.05262	0.26550	0.10160	0.26550	0.03000
Median	7015	50	13	45	39	7
Mode	7190	50	12	45	40	7
Standard Deviation	760.7927	1290.5262	2.6550	1.0160	2.6550	0.3000
Sample Variance	578805.5491	1665457.9672	7.0489	1.0322	7.0489	0.0900
Kurtosis	-0.1014	99.9013	0.2625	12.9307	0.2625	25.2722
Skewness	-0.1072	9.9927	0.3295	-3.3457	-0.3295	1.4524
Range	4245	12918	13	6	13	3
Minimum	4958	40	8	39	31	6
Maximum	9203	12958	21	45	44	9
Sum	688608	18485	1304	4459	3896	697
Count	100	100	100	100	100	100

Table 14. Descriptive statistics for Scenario 10 (x) and 3 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	4439.5	76.99	55.06	105.2	168.94	5.82
Standard Error	39.9874	3.6851	0.7445	4.0127	0.7445	0.1872
Median	4431.5	70	54	92	170	5
Mode	4126	60	52	213	172	5
Standard Deviation	399.8740	36.8512	7.4452	40.1273	7.4452	1.8715
Sample Variance	159899.2222	1358.0100	55.4307	1610.2020	55.4307	3.5026
Kurtosis	0.5263	19.0948	1.3384	1.4780	1.3384	1.5725
Skewness	0.3047	3.9944	0.2088	1.5754	-0.2088	1.4652
Range	2213	260	45	153	45	7
Minimum	3495	40	35	60	144	4
Maximum	5708	300	80	213	189	11
Sum	443950	7699	5506	10520	16894	582
Count	100	100	100	100	100	100

Table 15. Descriptive statistics for Scenario 11 (x) and 6 (y)

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	4670.21	78.61	20.4	25.66	31.6	3.54
Standard Error	55.65046	4.579023	0.298481	0.723099	0.298481	0.161696
Median	4616.5	70	20.5	23	31.5	3
Mode	4837	70	21	22	31	2
Standard Deviation	556.5046	45.79023	2.98481	7.230994	2.98481	1.616956
Sample Variance	309697.4	2096.745	8.909091	52.28727	8.909091	2.614545
Kurtosis	0.059125	10.04982	-0.15836	0.047472	-0.15836	-0.26892
Skewness	0.480137	3.192086	0.137677	0.859706	-0.13768	0.943365
Range	2765	260	15	30	15	5
Minimum	3615	20	14	15	23	2
Maximum	6380	280	29	45	38	7
Sum	467021	7861	2040	2566	3160	354
Count	100	100	100	100	100	100

Table 16. Descriptive statistics for Scenario 12 (x) and 8 (y)

Data Analysis

Hypothesis Test for Scenario 1 and 2

Table 17 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 1 and 2 assuming equal variances. Scenario 1 and 2 were represented by x and y, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	186.86	195.76	211.72	210.86	11	10.85
Variance	1830199	1836810	6.829899	12.84889	0.121212	0.391414
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	-0.04648		1.938649		2.095033	
P(*T*>=*t*) one-tail	0.481489		0.026983		0.01872	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 17. Hypothesis testing results for scenario 1 and 2

Analysis of the One-Tailed Test

- The CT of scenario 2 was not statistically significantly better than the CT of scenario 1 at the 0.05 level. The *P*-value for this test was 0.481489. There was not sufficient evidence to reject the null hypothesis.
- The *P*-value of the observed statistic for the TD one-tailed test between scenario 1 and 2 was 0.026983. Since this value was less than 0.05, then the TD of scenario 2 was judged statistically significantly better than the TD of scenario 1.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 1 and 2 was 0.01872. Again, this value was less than 0.05, therefore the FD of scenario 2 was judged statistically significantly better than the FD of scenario 1.

Previous analysis indicates that there was sufficient evidence to state that scenario 2 was statistically significantly better than scenario 1, suggesting scenario 2 as the best scenario to be used for comparison against scenario 9 (NSMutation).

Hypothesis Test for Scenario 3 and 4

Table 18 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 3 and 4, assuming equal variances. In this case, *x* and *y* represented scenario 3 and 4, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	189.77	184.85	44.5	44.59	6.9	6.97
Variance	1746232	1665458	0.858586	1.032222	0.151515	0.09
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	0.026637		-0.65451		-1.42438	
P(*T*>=*t*) one-tail	0.489388		0.25677		0.077955	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 18. Hypothesis testing results for scenario 3 and 4

Analysis of the One-Tailed Test

- The CT of scenario 4 was not statistically significantly better than the CT of scenario 3 at the 0.05 level. The *P*-value for this test was 0.489388. There was not sufficient evidence to reject the null hypothesis.
- The *P*-value of the observed statistic for the TD one-tailed test between scenario 3 and 4 was 0.25677. Since this value was not less than 0.05, the TD of scenario 4 was not judged statistically significantly better than the TD of scenario 3.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 3 and 4 was 0.077955. Given that this value was not less than 0.05, then there was

insufficient evidence to indicate that the FD of scenario 4 was better than the FD of scenario 3.

There were no statistically significant differences between any of the sample means of scenario 3 and 4; hence, any of these scenarios was judged acceptable for further comparison against scenario 10 (NSMutation). In this case, scenario 3 was selected.

Hypothesis Test for Scenario 5 and 6

Table 19 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 5 and 6, assuming equal variances. In this case, *x* and *y* represented scenario 5 and 6, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	82.08	76.99	184.71	105.2	10.02	5.82
Variance	839.7511	1358.01	114.5716	1610.202	0.807677	3.502626
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	1.085744		19.14501		20.22998	
P(*T*>=*t*) one-tail	0.139456		3.14E-47		2.25E-50	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 19. Hypothesis testing results for scenario 5 and 6

Analysis of the One-tailed Test

- There was not statistical significant difference between the CT of scenario 5 and 6 at the 0.05 level. The *P*-value for this test was 0.139456. The null hypothesis was not rejected.

- The *P*-value of the observed statistic for the TD one-tailed test between scenario 5 and 6 was less than 0.05; thus the TD of scenario 6 was judged to be significantly better than the TD of scenario 5.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 5 and 6 was less than 0.05; thus there was sufficient evidence to confirm that the FD of scenario 6 was better than the FD of scenario 5.

The preceding analysis stated that scenario 6 was statistically significantly better than scenario 5 and consequently convenient for a comparison against scenario 11 (NSMutation).

Hypothesis Test for Scenario 7 and 8

Table 20 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 7 and 8, assuming equal variances. In this case, scenario 7 and 8 were represented with *x* and *y*, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	87.33	78.61	35.32	25.66	5.49	3.54
Variance	1301.213	2096.745	11.77535	52.28727	1.161515	2.614545
Observations	100	100	100	100	100	100
Pooled Variance	1698.979		32.03131		1.88803	
Df	198		198		198	
t Stat	1.495917		12.0691		10.03495	
P(*T*>=*t*) one-tail	0.068134		8.48E-26		1.01E-19	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 20. Hypothesis testing results for scenario 7 and 8

Analysis of the One-tailed Test

- There was not sufficient evidence to reject the null hypothesis for the CT of scenario 7 and 8 at the 0.05 level. The *P*-value for this test was 0.068134.

- The *P*-value of the observed statistic for the TD one-tailed test between scenario 5 and 6 was less than 0.05; thus the TD of scenario 8 was judged to be significantly better than the TD of scenario 7.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 7 and 8 was less than 0.05; thus there was sufficient evidence to confirm that the FD of scenario 8 was better than the FD of scenario 7.

Previous analysis stated that scenario 8 was significantly better than scenario 7, and therefore, is appropriated for a comparison against scenario 12 (NSMutation).

Hypothesis Test for Scenario 2 and 6

Table 21 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 2 and 6, assuming equal variances. In this case, scenario 2 and 6 are represented by *x* and *y*, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	195.76	76.99	210.86	105.2	10.85	5.82
Variance	1836809.9822	1358.0100	12.8489	1610.2020	0.3914	3.5026
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	0.8760		26.2268		25.4899	
P(*T*>=*t*) one-tail	0.1910		1.2243E-66		9.5784E-65	
t Critical one-tail	1.6526		1.6526		1.6526	

Table 21. Hypothesis testing results for scenario 2 and 6

Analysis of the One-tailed Test

- The *P*-value of the observed statistic for the CT one-tailed test between scenario 2 and 6 was not less than 0.05; thus the CT of scenario 6 was not judged to be statistically significantly better than the CT of scenario 2.

- There is a statistically significant difference between the TD of scenario 2 and 6 at the 0.05 level. The null hypothesis was thus rejected.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 9 and 2 was less than 0.05; thus there was sufficient evidence to indicate that the FD of scenario 2 are better than the FD of scenario 9.

Preceding observations showed that the TD and FD of scenario 6 are statistically significantly better than the TD and FD of scenario 2. Figure 11 illustrates graphically that scenario 6 was better than scenario 2. The computation time was expressed in milliseconds.

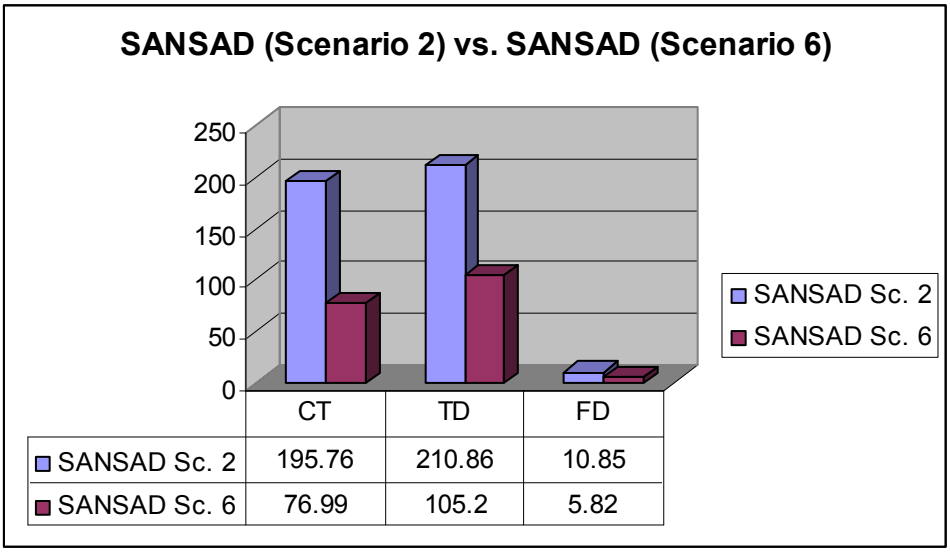


Figure 11. SANSAD (Scenario 2) vs. SANSAD (Scenario 6)

Hypothesis Test for Scenario 3 and 8

Table 22 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 3 and 8, assuming equal variances. In this case, scenario 3 and 8 were represented with x and y, respectively.

	CTx	*CTy*	*TDx*	*TDy*	*FDx*	*FDy*
Mean	189.77	78.61	44.5	25.66	6.9	3.54
Variance	1746232	2096.7454	0.8586	52.2873	0.1515	2.6145
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	0.8407		25.8432		20.2027	
P(T>=t) one-tail	0.2008		1.17E-65		2.7E-50	
t Critical one-tail	1.6526		1.6526		1.6526	

Table 22. Hypothesis testing results for scenario 3 and 8

Analysis of the One-tailed Test

- The *P*-value of the observed statistic for the CT one-tailed test between scenario 3 and 8 was not less than 0.05; thus the CT of scenario 8 was not judged statistically significantly better than the CT of scenario 3.
- There is a statistically significant difference between the TD of scenario 3 and 8 at the 0.05 level. The null hypothesis was thus rejected.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 3 and 8 was less than 0.05; thus there was sufficient evidence to indicate that the FD of scenario 8 was better than the FD of scenario 3.

The *P*-values generated by the one-tailed test for the TD and FD of scenario 8 were less than 0.05, indicating that scenario 8 was statistically significantly better than scenario 3. Figure 12 illustrates graphically that scenario 8 was better than scenario 3. The computation time was expressed in milliseconds.

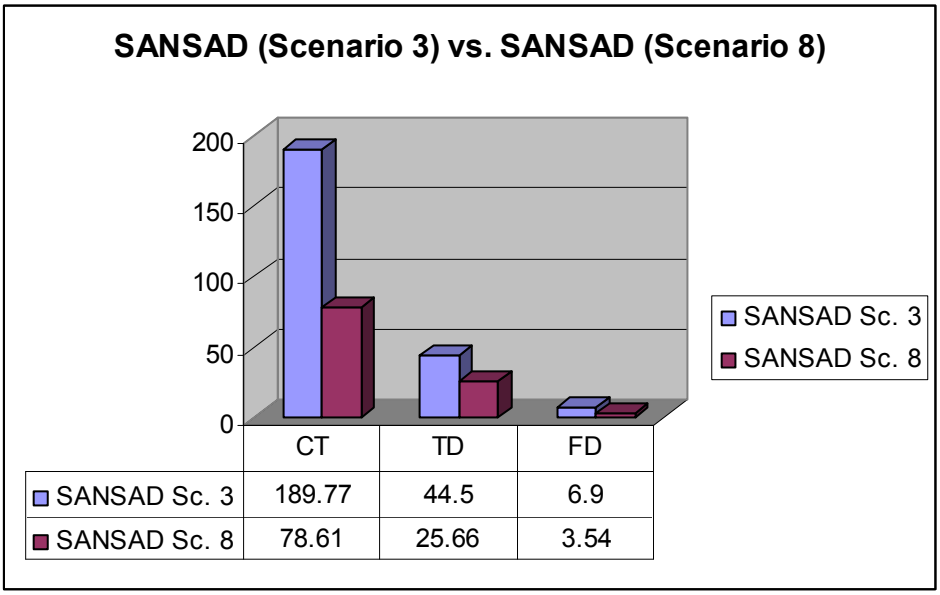


Figure 12. SANSAD (Scenario 3) vs. SANSAD (Scenario 8)

Hypothesis Test for Scenario 9 and 2

Table 23 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 9 and 2, assuming equal variances. In this case, scenario 9 and 2 were represented with *x* and *y*, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	6956.51	195.76	33.28	210.86	190.72	10.85
Variance	660274	1836810	15.65818	12.84889	15.65818	0.391414
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	42.7837		-332.597		448.9797	
P(T>=t) one-tail	2.7E-102		2.5E-274		4.4E-300	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 23. Hypothesis testing results for scenario 9 and 2

Analysis of the One-tailed Test

- The CT of scenario 9 and 2 presented a statistically significant difference at the 0.05 level. The null hypothesis was thus rejected.

- The *P*-value of the observed statistic for the TD one-tailed test between scenario 9 and 2 was less than 0.05; thus the TD of scenario 2 was judged to be statistically significantly better than the TD of scenario 9.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 9 and 2 was less than 0.05; thus there was sufficient evidence to indicate that the FD of scenario 2 was better than the FD of scenario 9.

The *P*-values generated by the one-tailed test for the CT, TD, and FD of scenario 2 and 9 were less than 0.05, indicating that scenario 2 was statistically significantly better than scenario 9. Figure 13 illustrates graphically the superiority of scenario 2 over scenario 9. The computation time was expressed in centiseconds. Figure 14 illustrates the computation time (expressed in centiseconds) and the percentage of TD and FD of NSMutation versus SANSAD.

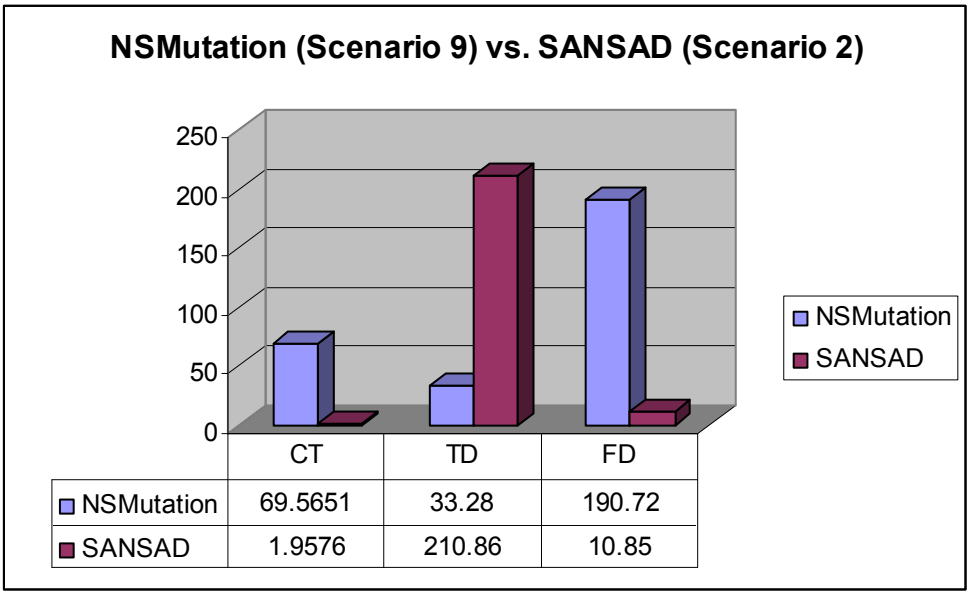


Figure 13. NSMutation (Scenario 9) vs. SANSAD (Scenario 2)

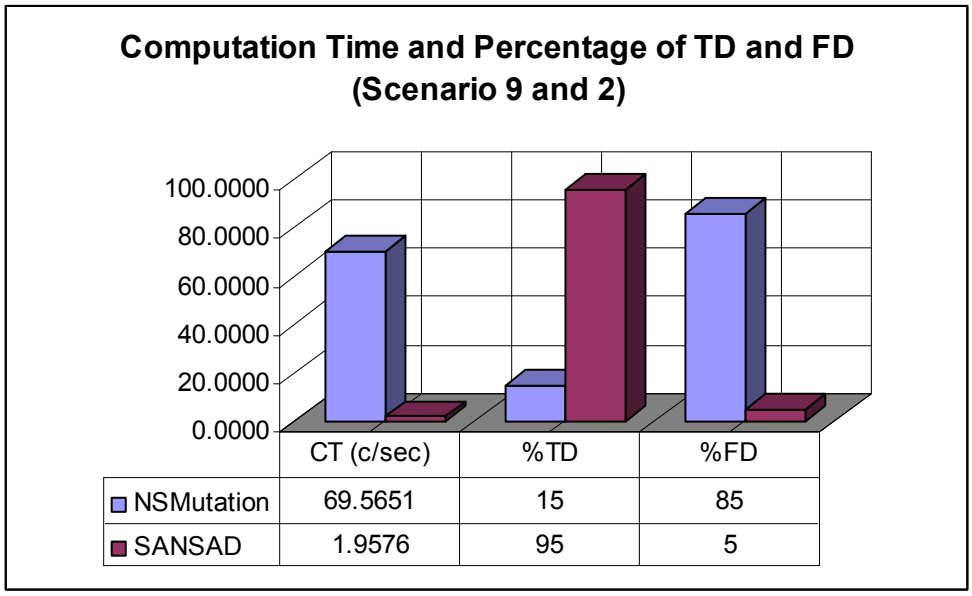


Figure 14. Computation time and percentage of TD and FD of NSMutation (Scenario 9) and SANSAD (Scenario 2)

Hypothesis Test for Scenario 10 and 3

Table 24 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 10 and 3, assuming equal variances. In this case, scenario 10 and 3 were represented with *x* and *y*, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	6886.08	184.85	13.04	44.59	38.96	6.97
Variance	578805.5	1665458	7.048889	1.032222	7.048889	0.09
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	44.73193		-110.985		119.7289	
P(T>=t) one-tail	9.2E-106		1.5E-180		5.6E-187	
t Critical one-tail	1.652586		1.652586		1.652586	

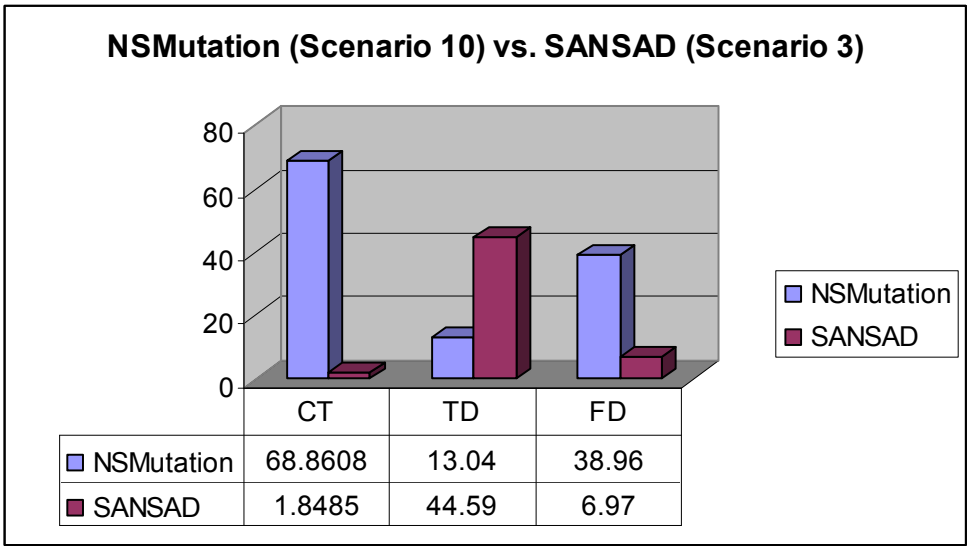
Table 24. Hypothesis testing results for scenario 10 and 3

Analysis of the One-tailed Test

- The CT of scenario 10 and 3 was statistically significant at the 0.05 level. The null hypothesis was thus rejected.

- The *P*-value of the observed statistic for the TD one-tailed test between scenario 10 and 3 was less than 0.05; thus the TD of scenario 3 was judged to be statistically significantly better than the TD of scenario 10.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 10 and 3 was less than 0.05; thus there was sufficient evidence to confirm that the FD of scenario 3 were better than the FD of scenario 10.

The *P*-values generated by the one-tailed test for the CT, TD, and FD of scenario 3 and 10 were less than 0.05, indicating that scenario 3 was statistically significantly better than scenario 10. Figure 15 illustrates graphically the supremacy of scenario 3 over scenario 10. The computation time was expressed in centiseconds. Figure 16 illustrates the computation time (expressed in centiseconds) and the percentage of TD and FD of NSMutation versus SANSAD.



	CT	TD	FD
NSMutation	68.8608	13.04	38.96
SANSAD	1.8485	44.59	6.97

Figure 15. NSMutation (Scenario 10) vs. SANSAD (Scenario 3)

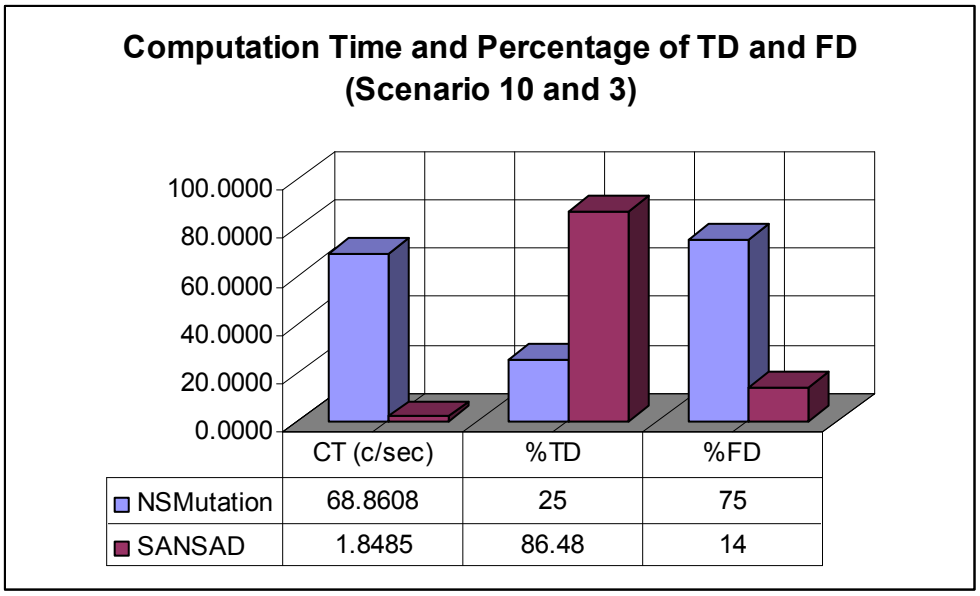


Figure 16. Computation time and percentage of TD and FD of NSMutation (Scenario 10) and SANSAD (Scenario 3)

Hypothesis Test for Scenario 11 and 6

Table 25 shows the *t*-test results for the two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 11 and 6, assuming equal variances. In this case, scenario 11 and 6 were represented with *x* and *y*, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	4439.5	76.99	55.06	105.2	168.94	5.82
Variance	159899.2	1358.01	55.43071	1610.202	55.43071	3.502626
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	108.6368		-12.2856		212.4842	
P(T>=t) one-tail	9.7E-179		1.86E-26		6.7E-236	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 25. Hypothesis testing results for scenario 11 and 6

Analysis of the One-tailed Test

- There was statistically significant difference between the CT of scenario 11 and 6 at the 0.05 level. The null hypothesis was thus rejected.

- The *P*-value of the observed statistic for the TD one-tailed test between scenario 11 and 6 was less than 0.05; thus the TD of scenario 6 was judged to be statistically significantly better than the TD of scenario 11.
- The *P*-value of the observed statistic for the FD one-tailed test between scenario 11 and 6 was less than 0.05; thus there was sufficient evidence to indicate that the FD of scenario 6 were better than the FD of scenario 11.

The *P*-values generated by the one-tailed test for the CT, TD and FD of scenario 6 and 13 were less than 0.05, indicating that scenario 6 was significantly better than scenario 11. Figure 17 illustrates graphically the supremacy of scenario 6 over scenario 11. The computation time was expressed in centiseconds. Figure 18 illustrates the computation time (expressed in centiseconds) and the percentage of TD and FD of NSMutation versus SANSAD.

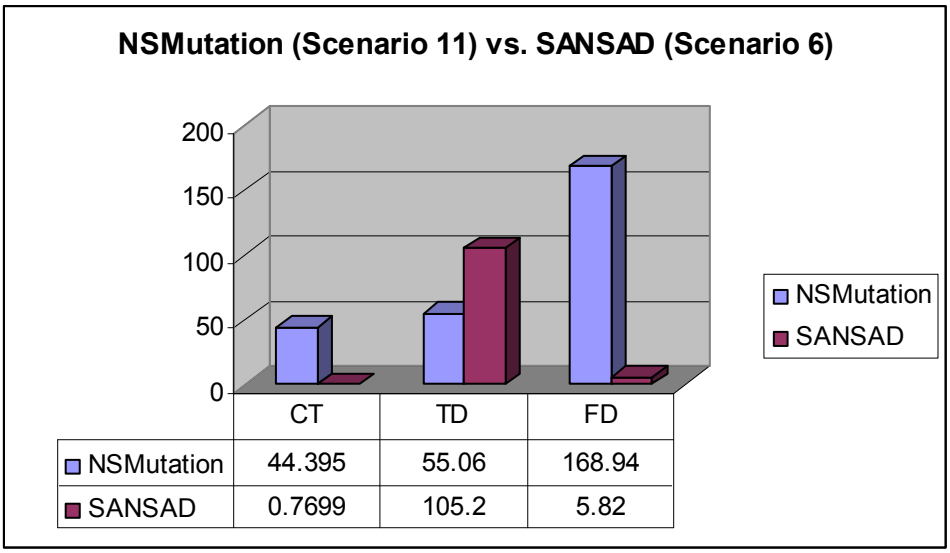


Figure 17. NSMutation (Scenario 11) vs. SANSAD (Scenario 6)

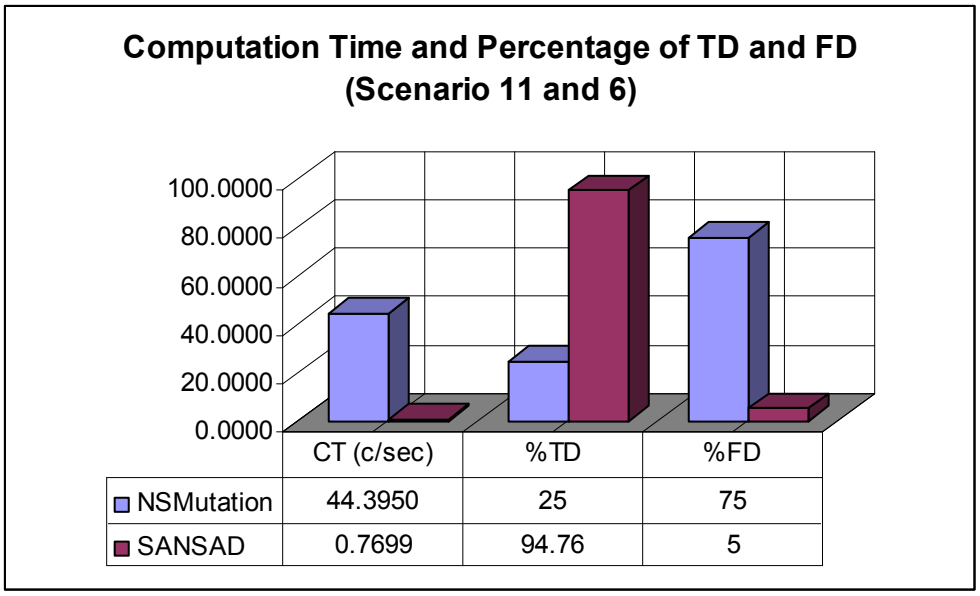


Figure 18. Computation time and percentage of TD and FD of NSMutation (Scenario 11) and SANSAD (Scenario 6)

Hypothesis Test for Scenario 12 and 8

Table 26 shows the *t*-test results for two independent sample means ($\bar{x}$ and $\bar{y}$) of scenario 12 and 8, assuming equal variances. In this case, scenario 12 and 8 were represented with *x* and *y*, respectively.

	CTx	***CTy***	***TDx***	***TDy***	***FDx***	***FDy***
Mean	4670.21	78.61	20.4	25.66	31.6	3.54
Variance	309697.4	2096.745	8.909091	52.28727	8.909091	2.614545
Observations	100	100	100	100	100	100
Df	198		198		198	
t Stat	82.22996		-6.72393		82.65952	
P(T>=t) one-tail	2.6E-155		9.24E-11		9.5E-156	
t Critical one-tail	1.652586		1.652586		1.652586	

Table 26. Hypothesis testing results for scenario 12 and 8

Analysis of the One-tailed Test

- There was statistically significant difference between the CT of scenario 8 and 12 at the 0.05 level. The null hypothesis was thus rejected.

- The *P*-value of the observed statistic for the TD one-tailed test between scenario 8 and 12 was less than 0.05; thus the TD of scenario 8 was judged statistically significantly better than the TD of scenario 12.
- The *P*-value of the observed statistic for the FD one-tailed test between scenarios 12 and 8 was less than 0.05; thus there was sufficient evidence to indicate that the FD of scenario 8 were better than the FD of scenario 12.

The *P*-values generated by the one-tailed test for the CT, TD, and FD of scenarios 8 and 12 were less than 0.05, indicating that scenario 8 was statistically significantly better than scenario 12. Figure 19 illustrates graphically the supremacy of scenario 8 over scenario 12. The computation time was expressed in centiseconds. Figure 20 illustrates the computation time (expressed in centiseconds) and the percentage of TD and FD of NSMutation versus SANSAD.

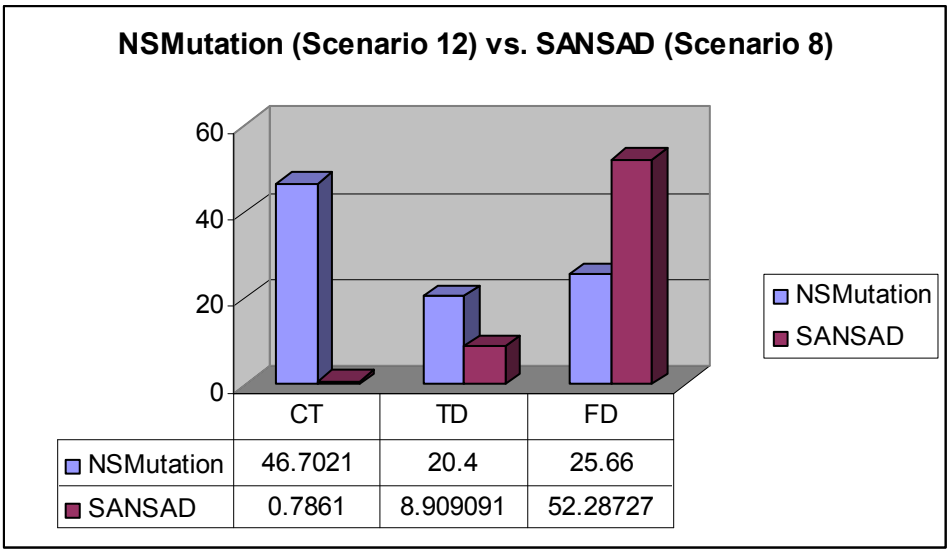


Figure 19. NSMutation (Scenario 12) vs. SANSAD (Scenario 8).

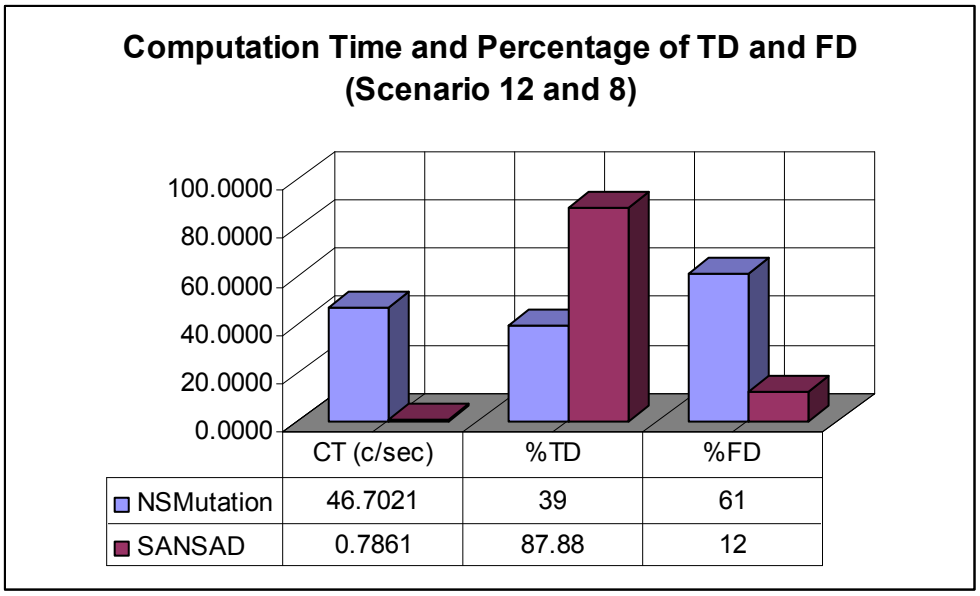


Figure 20. Computation time and percentage of TD and FD of NSMutation (Scenario 12) and SANSAD (Scenario 8)

Findings

- The TD and FD of scenarios 1 and 2 and scenarios 5 and 6 were statistically significantly different, with the results indicating that that the quality of the detectors generated by SANSAD were statistically better when *r*-contiguous bits matching rule together with recombination and mutation were used, regardless of the dataset size.
- There was no statistically significant difference between scenarios 3 and 4. Changing the variation operators when the Hamming distance matching rule was used did not appear to affect the quality of the detectors and the computation time of SANSAD.
- The TD and FD of scenarios 2 and 6 and scenarios 3 and 8 were statistically significantly different, which suggested that when a dataset contains more records (Net3 as compared to Net2), and the recombination and mutation operators were

used, regardless of the matching rule, the quality of the detectors generated by SANSAD were statistically better.

- The CT, TD, and FD of scenarios 9 and 2, scenarios 10 and 3, scenarios 11 and 6, and scenarios 12 and 8 were statistically significantly different, indicating that the quality of the detectors and the computation time of SANSAD were statistically better than the NSMutation, regardless of the dataset size and matching rule used.

Answers to the Research Questions

- What is the percentage of true detections of the detectors generated by the self-adaptive evolutionary negative selection algorithm, compared to the percentage of true detections of the detectors generated by the NSMutation algorithm?
    a. Scenario 9 (NSMutation) and 2 (SANSAD): the true detections of SANSAD were 540% greater than the true detections of the NSMutation.
    b. Scenario 10 (NSMutation) and 3 (SANSAD): the true detections of SANSAD were 245% greater than the true detections of the NSMutation.
    c. Scenario 11 (NSMutation) and 6 (SANSAD): the true detections of SANSAD were 286% greater than the true detections of the NSMutation.
    d. Scenario 12 (NSMutation) and 8 (SANSAD): the true detections of SANSAD were 124% greater than the true detections of the NSMutation.
- What is the percentage of false detections of the detectors generated by the self-adaptive evolutionary negative selection algorithm, compared to the percentage of false detections of the detectors generated by the NSMutation algorithm?
    a. Scenario 9 (NSMutation) and 2 (SANSAD): the false detections of SANSAD were 94% less than the false detections of the NSMutation.

b. Scenario 10 (NSMutation) and 3 (SANSAD): the false detections of SANSAD were 82% less than the false detections of the NSMutation.

c. Scenario 11 (NSMutation) and 6 (SANSAD): the false detections of SANSAD were 93% less than the false detections of the NSMutation.

d. Scenario 12 (NSMutation) and 8 (SANSAD): the false detections of SANSAD were 80% less than the false detections of the NSMutation.

- Is the novel self-adaptive evolutionary negative selection algorithm more efficient and efficient than the NSMutation algorithm?

a. Scenario 9 (NSMutation) and 2 (SANSAD): the computation time used by SANSAD was 97% less than the computation time used by the NSMutation algorithm.

b. Scenario 10 (NSMutation) and 3 (SANSAD): the computation time used by SANSAD was 98% less than the computation time used by the NSMutation algorithm.

c. Scenario 11 (NSMutation) and 6 (SANSAD): the computation time used by SANSAD was 152% less than the computation time used by the NSMutation algorithm.

d. Scenario 12 (NSMutation) and 8 (SANSAD): the computation time used by SANSAD was 145% less than the computation time used by the NSMutation algorithm.

The preceding list shows that the SANSAD algorithm was more competitive in terms of percentage of saved time, as well as more efficient.

- When and why is the novel self-adaptive evolutionary negative selection algorithm better than the NSMutation algorithm?
    a. SANSAD was superior to NSMutation in all the scenarios tested, which was attributable to the self-adaptation of the mutation step size.
- When and why is the novel self-adaptive evolutionary negative selection algorithm not better than the NSMutation algorithm?
    a. NSMutation did not outperform SANSAD in any tested scenario.

Summary of the Chapter

This chapter presented the results of the investigation. The data collected were organized and presented to reveal their meaning. Data resulting from the study were summarized in terms of their descriptive and inferential characteristics. Results and findings were derived from the analysis.

Chapter 5

Conclusion, Implications, Recommendations, and Summary

Conclusions

SANSAD outperformed NSMutation in all the scenarios tested regardless of the matching rule, variation operators, and dataset size. There was sufficient statistical evidence to suggest that SANSAD can generate detectors of better quality and with better computational time performance than the NSMutation algorithm by using self-adaptive techniques (dynamic parameter setting) to mutate the mutation step size of the detectors. In this case, quality refers to non-redundant detectors that maximize the true detections and minimizes the false detections. Additionally, the following conclusions were made:

- The computation time of SANSAD can be minimized and the quality of the detectors maximized when the *r*-contiguous bits matching rule is used together with recombination and mutation regardless of the dataset size.
- SANSAD performs better (from a computational time and quality of detectors perspective) when a dataset contains more records (Net3 versus Net2) using recombination and mutation as variation operators depicting the matching rule.

Implications

The flourishing of the Internet has changed the way many organizations and individuals do business. Before the popularity of the Internet, computers were limited to the walls of the organization where computers were linked to each other, but had little contact with computer systems outside the organization. Now, computers are exposed

and linked to the entire world, which has significantly improved the way data are exchanged. At the same time, privacy and assets have become vulnerable to innumerable attackers (hackers) who populate cyberspace. As a countermeasure to such risks, many organizations have implemented preventive techniques such as access control and authentication, and intrusion detection systems (IDS).

IDS essentially attempt to monitor and inspect all inbound and outbound network activity to detect suspicious patterns that may indicate an unauthorized presence on a network or system. IDS have alleviated many of the problems facing current network security; however, many of issues remain to be resolved, for example, a high dependency on having models of new attacks, and a high number of false negative and false positive detections. A false negative detection is intrusive behavior defined by the IDS as normal user behavior while a false positive detection is legitimate user behavior that is regarded by the IDS as intrusive behavior. The approach presented in this dissertation can help to resolve those issues: (1) SANSAD was inspired by the negative selection approach, which means that the system identifies any pattern of behavior different from that of the normal user without needing a predefined collection of known intrusion techniques or profiles; (2) The false detections (the sum of false negatives and false positives) are minimized. Additionally, as compared to other negative selection approaches, SANSAD does not require manual setting of mutation probabilities and generates detectors of quality with a competitive computational time.

Recommendations for Additional Studies

SANSAD uses binary representation. The reason for choosing this form of representation was based on its apparent simplicity to emulate the matching between

lymphocyte receptors and antigens in the immune system; however, when network activity is represented as a binary number, there are risks that the representation does not capture realistically the nature of the data. Hence, future research might be conducted to explore the performance of the algorithm using an exclusively real-valued representation. Furthermore, the use of other mechanisms of reproduction and variation operators can be explored to determine whether the computation time and quality of the detectors are affected either positive or negatively. For example, roulette wheel technique might be used for selecting parents, or crossover might be achieved by using other techniques, e.g., two-point crossover, uniform crossover or arithmetic crossover. Likewise, methods for reducing random fluctuations in mutative self-adaptation might be evaluated (Runarsson, 2002).

Summary of the Chapter

In this chapter, the results of the investigation were interpreted and qualified based on the analysis performed and results achieved. Alternative explanations for the findings were discussed. The implication section discussed the impact of the work on the field of study and its contribution to knowledge and professional practice. The recommendations section suggested the areas for further research.

Summary of the Dissertation

In Chapter 1, the time/size limitation of the process of generating the set of candidate detectors required by the negative selection was stated as the main reason for undertaking the dissertation effort. The linear, greedy, binary template, and the NSMutation approaches were introduced to establish the basis of the research presented

in this dissertation. The NSMutation was mentioned as an efficient algorithm that supports different matching rules and possesses tunable parameters. Since the detectors generated by the NSMutation are highly dependent on the strategy parameters values, which are not necessary optimal when they are determined manually, the use of self-adaptation to mutate the step size was considered as a possibility to maximize the number of true detections, and minimize the computation time and false detections. The reality that the effectiveness of an evolutionary algorithm depends on many of its components was explained within the barriers and issues section. The inability to randomly sample the entire population of computer network activity was mentioned as a limitation of the study. The experimentation of the self-adaptive evolutionary negative selection algorithm only in the context of computer security was stated as a delimitation of the study.

Chapter 2 established the context for the investigation beginning with an explanation of the immune system, following with an overview of artificial immune systems, continuing with an historical overview of the negative selection algorithm, and concluding with an outline of evolutionary algorithms and the self-adaptation approach. The immune system was defined as a network of different kinds of cells, B-lymphocytes and T-lymphocytes, which participate in the actual biological immune response. The mechanisms for the generation of B-lymphocytes and T-lymphocytes were explained to establish how can be used in different domains where the differentiation between self and non-self is required, e.g., pattern recognition and the security of information systems.

Evolutionary algorithms were described as a class of problem-solving methods inspired by the Darwinian theory on the origin of the species via natural selection. The

four most-common form of evolutionary algorithms were briefly explained, i.e., genetic algorithms, genetic programming, evolution strategies, and evolutionary programming. The chapter concluded by defining the self-adaptation approach as a mechanism that could alter dynamically the strategy parameters that control evolutionary algorithms, describing mutation and crossover operators as approaches for the self-adaptation of strategy parameters, and showing an analysis of the convergence properties of the algorithms of the form $(\mu + \lambda) - EA$ with elitist selection.

Chapter 3 delineated, in detail, the ways in which the research was conducted. Each step followed to implement the algorithm from a conceptual and technical perspective was indicated. Description of the datasets used for the experiments was provided, together with an explanation on how the data were migrated, selected, and profiled. The procedures followed for testing the self-adaptive evolutionary negative selection developed along with the NSMutation algorithm were explained. Conceptual and technical details of the design and implementation of SANSAD were listed. The projected outcomes of the research were delineated briefly. The numerical approach of the descriptive statistics was cited as a resource used to analyze the observations of computation time, and true and false detections for each set of detectors generated by trial of the self-adaptive evolutionary negative selection and the NSMutation algorithms. Hypothesis testing was mentioned as a technique used to evaluate the statistical difference between two sample means. Tables and bar graphs were cited as the formats used to present results.

In chapter 4, a proof that SANSAD is an instance of a $(\mu + \lambda) - EA$ with elitist selection was developed. The data collected were organized and presented to reveal their

meaning. This chapter included tables, figures, and charts. Findings were itemized based on results yielded by the hypothesis testing procedure.

Chapter 5 encompassed the results of the investigation, which were interpreted and qualified based on the analysis performed and results achieved. Alternative explanations for the findings were discussed. The impact of the work on the field of study and its contribution to knowledge and professional practice were discussed. Recommendations for further research were done.

Appendixes

Appendix A – Scripts for Creation of Tables

```
PROMPT Creating Table 'CLUSTER_ASSORTED'
CREATE TABLE CLUSTER_ASSORTED
 (ID NUMBER(15) NOT NULL
 ,CH_SHDR_ID NUMBER(15) NOT NULL
 ,CH_ID NUMBER(15) NOT NULL
 ,BINARY_STRING VARCHAR2(32) NOT NULL
 ,REAL_VALUE NUMBER(15) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER
 )
/

COMMENT ON COLUMN CLUSTER_ASSORTED.ID IS 'Cluster Assorted Unique Identifier'
/

COMMENT ON COLUMN CLUSTER_ASSORTED.CH_SHDR_ID IS 'Self dataset header unique identifier'
/

COMMENT ON COLUMN CLUSTER_ASSORTED.BINARY_STRING IS 'Binary Representation'
/

COMMENT ON COLUMN CLUSTER_ASSORTED.REAL_VALUE IS 'Real Value'
/

COMMENT ON COLUMN CLUSTER_ASSORTED.CREATION_DATE IS 'Creation Date'
/

PROMPT Creating Table 'CLUSTER_HEADER'
CREATE TABLE CLUSTER_HEADER
 (ID NUMBER(15) NOT NULL
 ,SHDR_ID NUMBER(15) NOT NULL
 ,DESCRIPTION VARCHAR2(32) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN CLUSTER_HEADER.SHDR_ID IS 'Self dataset header unique identifier'
/

PROMPT Creating Table 'SELF_HEADER'
CREATE TABLE SELF_HEADER
 (ID NUMBER(15) NOT NULL
 ,DESCRIPTION VARCHAR2(32) NOT NULL
 ,BINARY_STRING_LENGTH NUMBER(15) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN SELF_HEADER.ID IS 'Self dataset header unique identifier'
/

COMMENT ON COLUMN SELF_HEADER.DESCRIPTION IS 'Self dataset description'
```

```
/

COMMENT ON COLUMN SELF_HEADER.BINARY_STRING_LENGTH IS 'Self dataset binary string length'
/

PROMPT Creating Table 'SELF'
CREATE TABLE SELF
 (ID NUMBER(15) NOT NULL
 ,SHDR_ID NUMBER(15) NOT NULL
 ,BINARY_STRING VARCHAR2(32) NOT NULL
 ,REAL_VALUE NUMBER(15) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN SELF.ID IS 'Self unique identifier'
/

COMMENT ON COLUMN SELF.SHDR_ID IS 'Self dataset header unique identifier'
/

PROMPT Creating Table 'PORTS'
CREATE TABLE PORTS
 (PORT_NUMBER NUMBER(15) NOT NULL
 ,PROTOCOL VARCHAR2(5) NOT NULL
 ,KEYWORD VARCHAR2(32)
 ,DESCRIPTION VARCHAR2(64) NOT NULL
 ,RISK VARCHAR2(1)
 ,PORT_TYPE VARCHAR2(1) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN PORTS.PORT_NUMBER IS 'Port Number'
/

COMMENT ON COLUMN PORTS.PROTOCOL IS 'TCP / UDP'
/

COMMENT ON COLUMN PORTS.DESCRIPTION IS 'Description'
/

COMMENT ON COLUMN PORTS.RISK IS 'H / M / L'
/

COMMENT ON COLUMN PORTS.PORT_TYPE IS 'W / R / D'
/

PROMPT Creating Table 'SCENARIOS'
CREATE TABLE SCENARIOS
 (ID NUMBER(15) NOT NULL
 ,CH_SHDR_ID NUMBER(15) NOT NULL
 ,CH_ID NUMBER(15) NOT NULL
 ,MATCHING_PROBABILITY NUMBER(15,4) NOT NULL
 ,REPERTOIRE_SIZE NUMBER(15) NOT NULL
 ,PROBABILITY_FAILURE NUMBER(15,4) NOT NULL
 ,INITIAL_POPULATION NUMBER(15) NOT NULL
 ,MATCHING_THRESHOLD NUMBER(15) NOT NULL
 ,VARIATION_TYPE NUMBER(15) NOT NULL
 ,NUMBER_OF_TRIALS NUMBER(15) NOT NULL
 ,MATCHING_RULE VARCHAR2(1) NOT NULL
 ,APPROACH VARCHAR2(1) NOT NULL
 ,TOTAL_SELF NUMBER(15) NOT NULL
```

```
 ,TOTAL_CLUSTER NUMBER(15) NOT NULL
 ,TOTAL_NONSELF_IN_CLUSTER NUMBER(15)
 ,MEAN_TRUE_DETECTION NUMBER(15,4)
 ,STDEV_TRUE_DETECTION NUMBER(15,4)
 ,MEAN_FALSE_DETECTION NUMBER(15,4)
 ,STDEV_FALSE_DETECTION NUMBER(15,4)
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN SCENARIOS.ID IS 'Scenario Unique Identifier'
/

COMMENT ON COLUMN SCENARIOS.CH_SHDR_ID IS 'Self dataset header unique identifier'
/

COMMENT ON COLUMN SCENARIOS.VARIATION_TYPE IS 'Variation Type'
/

COMMENT ON COLUMN SCENARIOS.MATCHING_RULE IS 'R = R-Contiguous; H = Hamming Distance'
/

COMMENT ON COLUMN SCENARIOS.APPROACH IS 'A = Adaptive; S = Self Adaptive'
/

COMMENT ON COLUMN SCENARIOS.MEAN_TRUE_DETECTION IS 'Population Mean True Detections'
/

COMMENT ON COLUMN SCENARIOS.STDEV_TRUE_DETECTION IS 'Standard Deviation True Detections'
/

COMMENT ON COLUMN SCENARIOS.MEAN_FALSE_DETECTION IS 'Population Mean False Detections'
/

COMMENT ON COLUMN SCENARIOS.STDEV_FALSE_DETECTION IS 'Standard Deviation False
Detections'
/

PROMPT Creating Table 'TRIALS'
CREATE TABLE TRIALS
 (ID NUMBER(15) NOT NULL
 ,SCNR_CH_SHDR_ID NUMBER(15) NOT NULL
 ,SCNR_CH_ID NUMBER(15) NOT NULL
 ,SCNR_ID NUMBER(15) NOT NULL
 ,DURATION NUMBER(15,4)
 ,TOTAL_DETECTION NUMBER(15,4)
 ,TOTAL_TRUE_POSITIVE NUMBER(15,4)
 ,TOTAL_TRUE_NEGATIVE NUMBER(15,4)
 ,TOTAL_FALSE_POSITIVE NUMBER(15,4)
 ,TOTAL_FALSE_NEGATIVE NUMBER(15,4)
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN TRIALS.SCNR_CH_SHDR_ID IS 'Self dataset header unique identifier'
/

COMMENT ON COLUMN TRIALS.SCNR_ID IS 'Scenario Unique Identifier'
/

COMMENT ON COLUMN TRIALS.DURATION IS 'Trial Duration'
/
```

```
COMMENT ON COLUMN TRIALS.TOTAL_DETECTION IS 'Total Detection'
/

COMMENT ON COLUMN TRIALS.TOTAL_TRUE_POSITIVE IS 'True Positive'
/

COMMENT ON COLUMN TRIALS.TOTAL_TRUE_NEGATIVE IS 'True Negative'
/

COMMENT ON COLUMN TRIALS.TOTAL_FALSE_POSITIVE IS 'False Positive'
/

COMMENT ON COLUMN TRIALS.TOTAL_FALSE_NEGATIVE IS 'False Negative'
/

PROMPT Creating Table 'NONSELF'
CREATE TABLE NONSELF
 (ID NUMBER(15) NOT NULL
 ,TRIAL_SCNR_CH_SHDR_ID NUMBER(15) NOT NULL
 ,TRIAL_SCNR_CH_ID NUMBER(15) NOT NULL
 ,TRIAL_SCNR_ID NUMBER(15) NOT NULL
 ,TRIAL_ID NUMBER(15) NOT NULL
 ,BINARY_STRING VARCHAR2(32) NOT NULL
 ,REAL_VALUE NUMBER(15) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN NONSELF.TRIAL_SCNR_CH_SHDR_ID IS 'Self dataset header unique
identifier'
/

COMMENT ON COLUMN NONSELF.TRIAL_SCNR_ID IS 'Scenario Unique Identifier'
/

PROMPT Creating Table 'DETECTORS'
CREATE TABLE DETECTORS
 (ID NUMBER(15) NOT NULL
 ,TRIAL_SCNR_CH_SHDR_ID NUMBER(15) NOT NULL
 ,TRIAL_SCNR_CH_ID NUMBER(15) NOT NULL
 ,TRIAL_SCNR_ID NUMBER(15) NOT NULL
 ,TRIAL_ID NUMBER(15) NOT NULL
 ,BINARY_STRING VARCHAR2(32) NOT NULL
 ,REAL_VALUE NUMBER(15) NOT NULL
 ,MUTATION_STEP_SIZE NUMBER(15,4) NOT NULL
 ,AFFINITY NUMBER(32) NOT NULL
 ,CREATED_BY NUMBER(15)
 ,CREATION_DATE DATE
 ,LAST_UPDATED_BY NUMBER(15)
 ,LAST_UPDATE_DATE DATE
 ,LAST_UPDATE_LOGIN NUMBER(15)
 )
/

COMMENT ON COLUMN DETECTORS.ID IS 'Detector Unique Identifier'
/

COMMENT ON COLUMN DETECTORS.TRIAL_SCNR_CH_SHDR_ID IS 'Self dataset header unique
identifier'
/

COMMENT ON COLUMN DETECTORS.TRIAL_SCNR_ID IS 'Scenario Unique Identifier'
/

COMMENT ON COLUMN DETECTORS.BINARY_STRING IS 'Detector Binary Representation'
/
```

```
COMMENT ON COLUMN DETECTORS.REAL_VALUE IS 'Detector Real Value Representation'
/

COMMENT ON COLUMN DETECTORS.MUTATION_STEP_SIZE IS 'Mutation Step Size'
/

COMMENT ON COLUMN DETECTORS.AFFINITY IS 'Detector affinity'
/
```

Appendix B – Scripts for Creation of Indexes

```
PROMPT Creating Index 'CA_CH_FK_I'
CREATE INDEX CA_CH_FK_I ON CLUSTER_ASSORTED
 (CH_ID
 ,CH_SHDR_ID)
/

PROMPT Creating Index 'CH_SHDR_FK_I'
CREATE INDEX CH_SHDR_FK_I ON CLUSTER_HEADER
 (SHDR_ID)
/

PROMPT Creating Index 'SELF_SHDR_FK_I'
CREATE INDEX SELF_SHDR_FK_I ON SELF
 (SHDR_ID)
/

PROMPT Creating Index 'SCNR_CH_FK_I'
CREATE INDEX SCNR_CH_FK_I ON SCENARIOS
 (CH_ID
 ,CH_SHDR_ID)
/

PROMPT Creating Index 'TRIAL_SCNR_FK_I'
CREATE INDEX TRIAL_SCNR_FK_I ON TRIALS
 (SCNR_ID
 ,SCNR_CH_SHDR_ID
 ,SCNR_CH_ID)
/

PROMPT Creating Index 'NONSELF_TRIAL_FK_I'
CREATE INDEX NONSELF_TRIAL_FK_I ON NONSELF
 (TRIAL_ID
 ,TRIAL_SCNR_ID
 ,TRIAL_SCNR_CH_SHDR_ID
 ,TRIAL_SCNR_CH_ID)
/

PROMPT Creating Index 'DET_TRIAL_FK_I'
CREATE INDEX DET_TRIAL_FK_I ON DETECTORS
 (TRIAL_ID
 ,TRIAL_SCNR_ID
 ,TRIAL_SCNR_CH_SHDR_ID
 ,TRIAL_SCNR_CH_ID)
/
```

Appendix C – Scripts for Creation of Constraints

```
PROMPT Creating Primary Key on 'CLUSTER_ASSORTED'
ALTER TABLE CLUSTER_ASSORTED
 ADD (CONSTRAINT CA_PK PRIMARY KEY
  (ID
  ,CH_SHDR_ID
  ,CH_ID))
/

PROMPT Creating Primary Key on 'CLUSTER_HEADER'
ALTER TABLE CLUSTER_HEADER
 ADD (CONSTRAINT CH_PK PRIMARY KEY
  (ID
  ,SHDR_ID))
/

PROMPT Creating Primary Key on 'SELF_HEADER'
ALTER TABLE SELF_HEADER
 ADD (CONSTRAINT SHDR_PK PRIMARY KEY
  (ID))
/

PROMPT Creating Primary Key on 'SELF'
ALTER TABLE SELF
 ADD (CONSTRAINT SELF_PK PRIMARY KEY
  (ID
  ,SHDR_ID))
/

PROMPT Creating Primary Key on 'PORTS'
ALTER TABLE PORTS
 ADD (CONSTRAINT PORT_PK PRIMARY KEY
  (PORT_NUMBER
  ,PROTOCOL))
/

PROMPT Creating Primary Key on 'SCENARIOS'
ALTER TABLE SCENARIOS
 ADD (CONSTRAINT SCNR_PK PRIMARY KEY
  (ID
  ,CH_SHDR_ID
  ,CH_ID))
/

PROMPT Creating Primary Key on 'TRIALS'
ALTER TABLE TRIALS
 ADD (CONSTRAINT TRIAL_PK PRIMARY KEY
  (ID
  ,SCNR_CH_SHDR_ID
  ,SCNR_CH_ID
  ,SCNR_ID))
/

PROMPT Creating Primary Key on 'NONSELF'
ALTER TABLE NONSELF
 ADD (CONSTRAINT NONSELF_PK PRIMARY KEY
  (ID
  ,TRIAL_SCNR_CH_SHDR_ID
  ,TRIAL_SCNR_CH_ID
  ,TRIAL_SCNR_ID
  ,TRIAL_ID))
/

PROMPT Creating Primary Key on 'DETECTORS'
ALTER TABLE DETECTORS
 ADD (CONSTRAINT DET_PK PRIMARY KEY
  (ID
```

```
  ,TRIAL_SCNR_CH_SHDR_ID
  ,TRIAL_SCNR_CH_ID
  ,TRIAL_SCNR_ID
  ,TRIAL_ID))
/



PROMPT Creating Foreign Key on 'CLUSTER_ASSORTED'
ALTER TABLE CLUSTER_ASSORTED ADD (CONSTRAINT
 CA_CH_FK FOREIGN KEY
  (CH_ID
  ,CH_SHDR_ID) REFERENCES CLUSTER_HEADER
  (ID
  ,SHDR_ID))
/

PROMPT Creating Foreign Key on 'CLUSTER_HEADER'
ALTER TABLE CLUSTER_HEADER ADD (CONSTRAINT
 CH_SHDR_FK FOREIGN KEY
  (SHDR_ID) REFERENCES SELF_HEADER
  (ID))
/

PROMPT Creating Foreign Key on 'SELF'
ALTER TABLE SELF ADD (CONSTRAINT
 SELF_SHDR_FK FOREIGN KEY
  (SHDR_ID) REFERENCES SELF_HEADER
  (ID))
/

PROMPT Creating Foreign Key on 'SCENARIOS'
ALTER TABLE SCENARIOS ADD (CONSTRAINT
 SCNR_CH_FK FOREIGN KEY
  (CH_ID
  ,CH_SHDR_ID) REFERENCES CLUSTER_HEADER
  (ID
  ,SHDR_ID))
/

PROMPT Creating Foreign Key on 'TRIALS'
ALTER TABLE TRIALS ADD (CONSTRAINT
 TRIAL_SCNR_FK FOREIGN KEY
  (SCNR_ID
  ,SCNR_CH_SHDR_ID
  ,SCNR_CH_ID) REFERENCES SCENARIOS
  (ID
  ,CH_SHDR_ID
  ,CH_ID))
/

PROMPT Creating Foreign Key on 'NONSELF'
ALTER TABLE NONSELF ADD (CONSTRAINT
 NONSELF_TRIAL_FK FOREIGN KEY
  (TRIAL_ID
  ,TRIAL_SCNR_ID
  ,TRIAL_SCNR_CH_SHDR_ID
  ,TRIAL_SCNR_CH_ID) REFERENCES TRIALS
  (ID
  ,SCNR_ID
  ,SCNR_CH_SHDR_ID
  ,SCNR_CH_ID))
/

PROMPT Creating Foreign Key on 'DETECTORS'
ALTER TABLE DETECTORS ADD (CONSTRAINT
 DET_TRIAL_FK FOREIGN KEY
  (TRIAL_ID
  ,TRIAL_SCNR_ID
  ,TRIAL_SCNR_CH_SHDR_ID
```

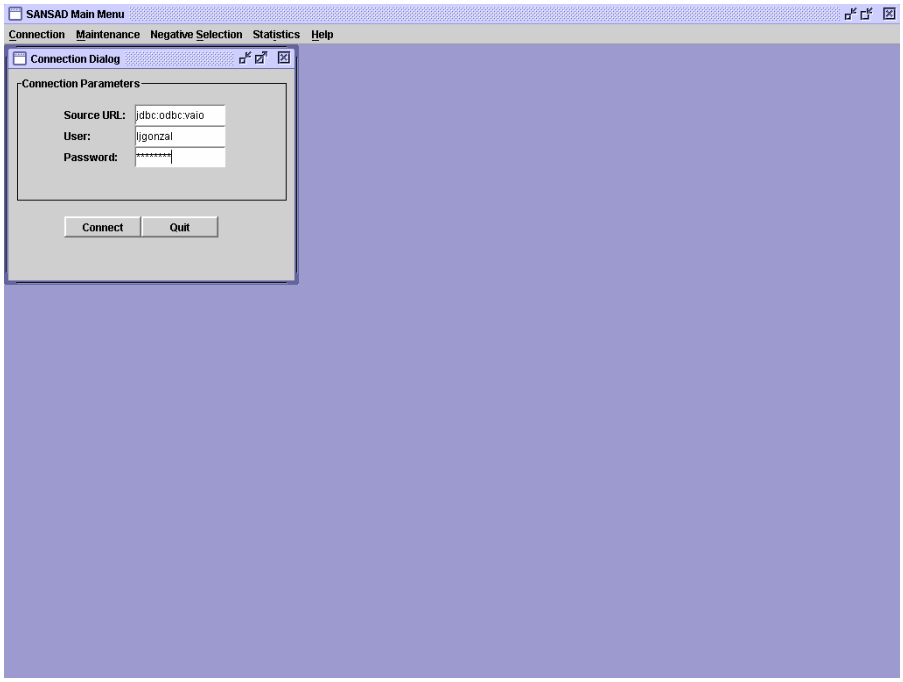
```
  ,TRIAL_SCNR_CH_ID) REFERENCES TRIALS
  (ID
  ,SCNR_ID
  ,SCNR_CH_SHDR_ID
  ,SCNR_CH_ID))
/
```

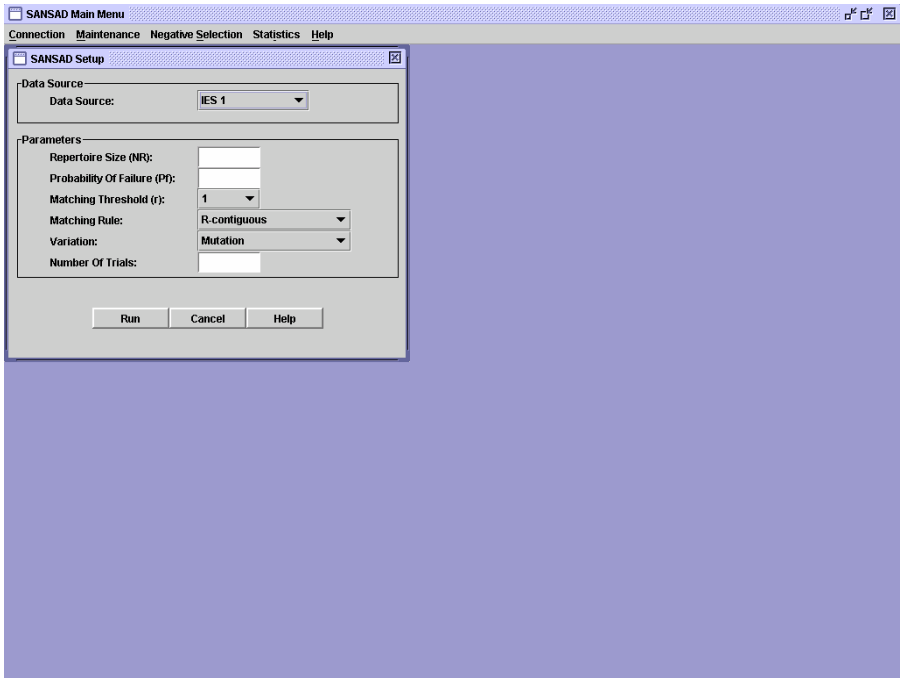
Appendix D – Scripts for Creation of Views

```
CREATE OR REPLACE FORCE VIEW V_UNIQUE_BASE_OUT_WITHOUT_80
(ID, SHDR_ID, REAL_VALUE)
AS
(SELECT ROWNUM id, '2' shdr_id, real_value
      FROM (SELECT DISTINCT TO_NUMBER (
                                 t1.src_addr
                              || LPAD (t1.dest_addr, 2, '0')
                                                  || LPAD (t1.dest_port, 4, '0')
                           ) real_value
                       FROM dataset_interface t1
                      WHERE t1.dataset_name = 'BASE'
                        AND t1.src_addr NOT IN (SELECT t2.self_addr
                                                  FROM self_addr t2
                                                 WHERE t2.self_addr = t1.src_addr)
                        AND t1.src_port <> '80'));


CREATE OR REPLACE FORCE VIEW V_UNIQUE_NET1_OUT_WITHOUT_80
(ID, CH_SHDR_ID, CH_ID, REAL_VALUE)
AS
(SELECT ROWNUM id, '2' ch_shdr_id, '5' ch_id, real_value
      FROM (SELECT DISTINCT TO_NUMBER (
                                 t1.src_addr
                              || LPAD (t1.dest_addr, 2, '0')
                              || LPAD (t1.dest_port, 4, '0')
                           ) real_value
                       FROM dataset_interface t1
                      WHERE t1.dataset_name = 'NET1'
                        AND t1.src_addr NOT IN (SELECT t2.self_addr
                                                  FROM self_addr t2
                                                 WHERE t2.self_addr = t1.src_addr)
                        AND t1.src_port <> '80'));


CREATE OR REPLACE FORCE VIEW V_UNIQUE_NET2_OUT_WITHOUT_80
(ID, CH_SHDR_ID, CH_ID, REAL_VALUE)
AS
(SELECT ROWNUM id, '2' ch_shdr_id, '6' ch_id, real_value
      FROM (SELECT DISTINCT TO_NUMBER (
                                 t1.src_addr
                              || LPAD (t1.dest_addr, 2, '0')
                              || LPAD (t1.dest_port, 4, '0')
                           ) real_value
                       FROM dataset_interface t1
                      WHERE t1.dataset_name = 'NET2'
                        AND t1.src_addr NOT IN (SELECT t2.self_addr
                                                  FROM self_addr t2
                                                 WHERE t2.self_addr = t1.src_addr)
                        AND t1.src_port <> '80'));


CREATE OR REPLACE FORCE VIEW V_UNIQUE_NET3_OUT_WITHOUT_80
(ID, CH_SHDR_ID, CH_ID, REAL_VALUE)
AS
(SELECT ROWNUM id, '2' ch_shdr_id, '7' ch_id, real_value
      FROM (SELECT DISTINCT TO_NUMBER (
                                 t1.src_addr
                              || LPAD (t1.dest_addr, 2, '0')
                              || LPAD (t1.dest_port, 4, '0')
                           ) real_value
                       FROM dataset_interface t1
                      WHERE t1.dataset_name = 'NET3'
                        AND t1.src_addr NOT IN (SELECT t2.self_addr
                                                  FROM self_addr t2
                                                 WHERE t2.self_addr = t1.src_addr)
                        AND t1.src_port <> '80'));
```

```
CREATE OR REPLACE FORCE VIEW V_UNIQUE_NET4_OUT_WITHOUT_80
(ID, CH_SHDR_ID, CH_ID, REAL_VALUE)
AS
(SELECT ROWNUM id, '2' ch_shdr_id, '8' ch_id, real_value
      FROM (SELECT DISTINCT TO_NUMBER (
                                  t1.src_addr
                               || LPAD (t1.dest_addr, 2, '0')
                               || LPAD (t1.dest_port, 4, '0')
                            ) real_value
                       FROM dataset_interface t1
                      WHERE t1.dataset_name = 'NET4'
                        AND t1.src_addr NOT IN (SELECT t2.self_addr
                                                  FROM self_addr t2
                                                 WHERE t2.self_addr = t1.src_addr)
                        AND t1.src_port <> '80'));
```

Appendix E – Screenshots of SANSAD





Appendix F – Java Source Code for MainMenu.java

```
/*
 * @(#) MainMenu.java 1.00 04/26/04
 *
 * Copyright © 2004 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */

import javax.swing.JOptionPane;
import javax.swing.JInternalFrame;
import javax.swing.JDesktopPane;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JFrame;
import javax.swing.KeyStroke;
import java.awt.event.*;
import java.awt.*;


/** MainMenu class
 */

/**
 * @author     Luis Gonzalez
 * @version    1.4.2
 */
public class MainMenu extends JFrame implements ActionListener {
    JDesktopPane desktop;
    public MainMenu() {
        super("SANSAD Main Menu");

        // Make the big window be indented 50 pixels from each edge
        // of the screen.

        int inset = 50;
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

        setBounds(inset, inset, screenSize.width - inset * 2,
                screenSize.height - inset * 2);

        // Set up the GUI.
        desktop = new JDesktopPane(); // a specialized layered pane

        createConnectionDialog(); // create first "window"
        setContentPane(desktop);
        setJMenuBar(createMenuBar());

        // Make dragging a little faster but perhaps uglier.
        desktop.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
    }

    protected JMenuBar createMenuBar() {
        JMenuBar menuBar = new JMenuBar();

        // Set up the lone menu.
        JMenu menu = new JMenu("Connection");

        menu.setMnemonic(KeyEvent.VK_C);
        menuBar.add(menu);

        // Set up the first menu item.
        JMenuItem menuItem = new JMenuItem("New");

        menuItem.setMnemonic(KeyEvent.VK_N);
        menuItem.setAccelerator(
```

```
        KeyStroke.getKeyStroke(KeyEvent.VK_N, ActionEvent.ALT_MASK));
menuItem.setActionCommand("connection");
menuItem.addActionListener(this);
menu.add(menuItem);

// Set up the second menu item.
menuItem = new JMenuItem("Quit");
menuItem.setMnemonic(KeyEvent.VK_Q);
menuItem.setAccelerator(
        KeyStroke.getKeyStroke(KeyEvent.VK_Q, ActionEvent.ALT_MASK));
menuItem.setActionCommand("quit");
menuItem.addActionListener(this);
menu.add(menuItem);

//

// Set up the lone menu.
JMenu menuMaintenance = new JMenu("Maintenance");

menuMaintenance.setMnemonic(KeyEvent.VK_M);
menuBar.add(menuMaintenance);

// Set up the first menu item.
JMenuItem menuDeleteTables = new JMenuItem("Delete Tables");

menuDeleteTables.setMnemonic(KeyEvent.VK_D);
menuDeleteTables.setAccelerator(
        KeyStroke.getKeyStroke(KeyEvent.VK_D, ActionEvent.ALT_MASK));
menuDeleteTables.setActionCommand("deleteTables");
menuDeleteTables.addActionListener(this);
menuMaintenance.add(menuDeleteTables);

// Set up the lone menu.
JMenu menuNegativeSelection = new JMenu("Negative Selection");

menuNegativeSelection.setMnemonic(KeyEvent.VK_S);
menuBar.add(menuNegativeSelection);

// Set up the first menu item.
JMenuItem menuGenerateDetectors = new JMenuItem("Generate Detectors");

menuGenerateDetectors.setMnemonic(KeyEvent.VK_G);
menuGenerateDetectors.setAccelerator(
        KeyStroke.getKeyStroke(KeyEvent.VK_G, ActionEvent.ALT_MASK));
menuGenerateDetectors.setActionCommand("generateDetectors");
menuGenerateDetectors.addActionListener(this);
menuNegativeSelection.add(menuGenerateDetectors);

// Set up the lone menu.
JMenu menuStatistics = new JMenu("Statistics");

menuStatistics.setMnemonic(KeyEvent.VK_I);
menuBar.add(menuStatistics);

// Set up the first menu item.
JMenuItem menuStatisticsByScenario = new JMenuItem("By Scenario");

menuStatisticsByScenario.setMnemonic(KeyEvent.VK_E);
menuStatisticsByScenario.setAccelerator(
        KeyStroke.getKeyStroke(KeyEvent.VK_E, ActionEvent.ALT_MASK));
menuStatisticsByScenario.setActionCommand("statisticsByScenario");
menuStatisticsByScenario.addActionListener(this);
menuStatistics.add(menuStatisticsByScenario);

// Set up the second menu item.
JMenuItem menuStatisticsByTrial = new JMenuItem("By Trial");

menuStatisticsByTrial.setMnemonic(KeyEvent.VK_T);
menuStatisticsByTrial.setAccelerator(
        KeyStroke.getKeyStroke(KeyEvent.VK_T, ActionEvent.ALT_MASK));
menuStatisticsByTrial.setActionCommand("statisticsByTrial");
```

```
        menuStatisticsByTrial.addActionListener(this);
        menuStatistics.add(menuStatisticsByTrial);

        // Set up the lone menu.
        JMenu menuHelp = new JMenu("Help");

        menuHelp.setMnemonic(KeyEvent.VK_H);
        menuBar.add(menuHelp);

        // Set up the first menu item.
        JMenuItem menuHelpAbout = new JMenuItem("About");

        menuHelpAbout.setMnemonic(KeyEvent.VK_A);
        menuHelpAbout.setAccelerator(
                KeyStroke.getKeyStroke(KeyEvent.VK_A, ActionEvent.ALT_MASK));
        menuHelpAbout.setActionCommand("helpAbout");
        menuHelpAbout.addActionListener(this);
        menuHelp.add(menuHelpAbout);

        //
        return menuBar;
    }

    // React to menu selections.
    public void actionPerformed(ActionEvent e) {
        if ("connection".equals(e.getActionCommand())) { // connection
            createConnectionDialog();
        } else if ("new".equals(e.getActionCommand())) { // new
            createAlgorithmFrame();
        } else if ("quit".equals(e.getActionCommand())) { // quit
            quit();
        } else if ("deleteTables".equals(e.getActionCommand())) { // deleteTables
            createDeleteTablesDialog();
        } else if ("generateDetectors".equals(e.getActionCommand())) { //
generateDetectors
            createAlgorithmFrame();
        } else if ("statisticsByScenario".equals(e.getActionCommand())) { //
statisticsByScenario
            createStatisticsByScenario();
        } else if ("statisticsByTrial".equals(e.getActionCommand())) { //
statisticsByTrial
            createStatisticsByTrial();
        } else if ("helpAbout".equals(e.getActionCommand())) { // helpAbout
            createAbout();
        }
    }

    // Create connection dialog.
    protected void createConnectionDialog() {
        ConnectionDialog frame = new ConnectionDialog();

        frame.setVisible(true); // necessary as of 1.3

        desktop.add(frame);
        try {
            frame.setSelected(true);
        } catch (java.beans.PropertyVetoException e) {}

    }

    // Create delete tables dialog.
    protected void createDeleteTablesDialog() {
        JOptionPane.showConfirmDialog(null,
                "All the activity will be irreversibly deleted from the database.\n"
                + "Do you want to continue?",
                "Warning",
                JOptionPane.YES_NO_OPTION);
    }

    // Create a new internal frame.
    protected void createAlgorithmFrame() {
```

```
        AlgorithmFrame frame = new AlgorithmFrame();

        frame.setVisible(true); // necessary as of 1.3
        desktop.add(frame);
        try {
            frame.setSelected(true);
        } catch (java.beans.PropertyVetoException e) {}
    }

    // Create statistics by scenario frame.
    protected void createStatisticsByScenario() {

        JOptionPane.showMessageDialog(null,
                "Statistics by scenario are not available", "Information",
                JOptionPane.INFORMATION_MESSAGE);

    }

    // Create statistics by trial frame.
    protected void createStatisticsByTrial() {
        JOptionPane.showMessageDialog(null,
                "Statistics by trial are not available", "Information",
                JOptionPane.INFORMATION_MESSAGE);
    }

    // Create an about internal frame.
    protected void createAbout() {
        JOptionPane.showMessageDialog(null,
                "A self-adaptive negative selection approach for anomaly detection\n"
                + "Copyright 2004 Luis J. Gonzalez \n" + "\n"
                + "SANSAD - Version 1.0",
                "About",
                JOptionPane.INFORMATION_MESSAGE);
    }

    // Quit the application.
    protected void quit() {
        System.exit(0);
    }

    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {

        // Make sure we have nice window decorations.
        JFrame.setDefaultLookAndFeelDecorated(true);

        // Create and set up the window.
        MainMenu frame = new MainMenu();

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Display the window.
        frame.setVisible(true);
    }

    public static void main(String[] args) {

        // Schedule a job for the event-dispatching thread:

        // creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Appendix G – Java Source Code for AlgorithmFrame.java

```
/*
 * @(#) AlgorithmFrame.java 1.00 04/24/04
 *
 * Copyright © 2004 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.text.*;
import java.sql.Connection;


// public class AlgorithmFrame extends JFrame {
public class AlgorithmFrame extends JInternalFrame {

    JLabel labDataSource;
    JComboBox comDataSource;
    JLabel labRepertoireSize;
    JFormattedTextField tfRepertoireSize;
    JLabel labProbOfNotMatchingSelf;
    JFormattedTextField tfProbOfNotMatchingSelf;
    JLabel labMatchingThreshold;
    JComboBox comMatchingThreshold;
    JLabel labMatchingRule;
    JComboBox comMatchingRule;
    JLabel labVariation;
    JComboBox comVariation;
    JLabel labNumberOfTrials;
    JFormattedTextField tfNumberOfTrials;
    JButton btnRun;
    JButton btnCancel;
    JButton btnHelp;
    JPanel panDataSource;
    JPanel panParameters;
    NumberFormat formatInteger = NumberFormat.getIntegerInstance();
    NumberFormat formatNumber = NumberFormat.getNumberInstance();

    public AlgorithmFrame() {
        super("SANSAD Setup", true, true, false, false);
        AlgorithmFrameLayout customLayout = new AlgorithmFrameLayout();

        getContentPane().setFont(new Font("Helvetica", Font.PLAIN, 12));
        getContentPane().setLayout(customLayout);

        labDataSource = new JLabel("Data Source:");
        getContentPane().add(labDataSource);

        comDataSource = new JComboBox();
        comDataSource.addItem("IES 1");
        comDataSource.addItem("IES 2");
        comDataSource.addItem("IES 3");
        comDataSource.addItem("IES 4");
        comDataSource.addItem("IES C");
        comDataSource.addItem("Primes 1");
        comDataSource.addItem("Primes 2");
        comDataSource.addItem("Primes 3");
        comDataSource.addItem("Primes 4");
        getContentPane().add(comDataSource);

        labRepertoireSize = new JLabel("Repertoire Size (NR):");
        getContentPane().add(labRepertoireSize);
```

```
tfRepertoireSize = new JFormattedTextField(formatInteger);
tfRepertoireSize.setBorder(new BevelBorder(BevelBorder.LOWERED));

getContentPane().add(tfRepertoireSize);

labProbOfNotMatchingSelf = new JLabel("Probability Of Failure (Pf):");
getContentPane().add(labProbOfNotMatchingSelf);

tfProbOfNotMatchingSelf = new JFormattedTextField(formatNumber);
tfProbOfNotMatchingSelf.setBorder(new BevelBorder(BevelBorder.LOWERED));

getContentPane().add(tfProbOfNotMatchingSelf);

labMatchingThreshold = new JLabel("Matching Threshold (r):");
getContentPane().add(labMatchingThreshold);

comMatchingThreshold = new JComboBox();
comMatchingThreshold.addItem("1");
comMatchingThreshold.addItem("2");
comMatchingThreshold.addItem("3");
comMatchingThreshold.addItem("4");
comMatchingThreshold.addItem("5");
comMatchingThreshold.addItem("6");
comMatchingThreshold.addItem("7");
comMatchingThreshold.addItem("8");
comMatchingThreshold.addItem("9");
comMatchingThreshold.addItem("10");
comMatchingThreshold.addItem("11");
comMatchingThreshold.addItem("12");
comMatchingThreshold.addItem("13");
comMatchingThreshold.addItem("14");
comMatchingThreshold.addItem("15");
comMatchingThreshold.addItem("16");
comMatchingThreshold.addItem("17");
comMatchingThreshold.addItem("18");
comMatchingThreshold.addItem("19");
comMatchingThreshold.addItem("20");
comMatchingThreshold.addItem("21");
comMatchingThreshold.addItem("22");
comMatchingThreshold.addItem("23");
comMatchingThreshold.addItem("24");
comMatchingThreshold.addItem("25");
comMatchingThreshold.addItem("26");
comMatchingThreshold.addItem("27");
comMatchingThreshold.addItem("28");
comMatchingThreshold.addItem("29");
comMatchingThreshold.addItem("30");
comMatchingThreshold.addItem("31");
comMatchingThreshold.addItem("32");
getContentPane().add(comMatchingThreshold);

labMatchingRule = new JLabel("Matching Rule:");
getContentPane().add(labMatchingRule);

comMatchingRule = new JComboBox();
comMatchingRule.addItem("R-contiguous");
comMatchingRule.addItem("Hamming Distance");
getContentPane().add(comMatchingRule);

labVariation = new JLabel("Variation:");
getContentPane().add(labVariation);

comVariation = new JComboBox();
comVariation.addItem("Mutation");
comVariation.addItem("Recombination & Mutation");
getContentPane().add(comVariation);

labNumberOfTrials = new JLabel("Number Of Trials:");
getContentPane().add(labNumberOfTrials);

tfNumberOfTrials = new JFormattedTextField(formatInteger);
```

```
    tfNumberOfTrials.setBorder(new BevelBorder(BevelBorder.LOWERED));
    getContentPane().add(tfNumberOfTrials);

    btnRun = new JButton("Run");
    btnRun.setBorder(new BevelBorder(BevelBorder.RAISED));
    getContentPane().add(btnRun);
    btnRun.setActionCommand("run");
    // btnRun.addActionListener(this);
    btnRun.addActionListener(new PushButtonActionListener());
    getContentPane().add(btnRun);

    btnCancel = new JButton("Cancel");
    btnCancel.setBorder(new BevelBorder(BevelBorder.RAISED));
    getContentPane().add(btnCancel);
    btnCancel.setActionCommand("cancel");
    // btnCancel.addActionListener(this);
    btnCancel.addActionListener(new PushButtonActionListener());
    getContentPane().add(btnCancel);

    btnHelp = new JButton("Help");
    btnHelp.setBorder(new BevelBorder(BevelBorder.RAISED));
    getContentPane().add(btnHelp);
    btnHelp.setActionCommand("help");
    // btnHelp.addActionListener(this);
    btnHelp.addActionListener(new PushButtonActionListener());
    getContentPane().add(btnHelp);

    panDataSource = new JPanel();
    panDataSource.setBorder(
            new TitledBorder(new LineBorder(Color.black, 1), "Data Source"));
    getContentPane().add(panDataSource);

    panParameters = new JPanel();
    panParameters.setBorder(
            new TitledBorder(new LineBorder(Color.black, 1), "Parameters"));
    getContentPane().add(panParameters);

    setSize(getPreferredSize());

    /* addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent e) {
     System.exit(0);
     }
     });
     */
}

/* public static void main(String args[]) {
 AlgorithmFrame window = new AlgorithmFrame();

 window.setTitle("AlgorithmFrame");
 window.pack();
 window.show();
 }

 }
 */

private class PushButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent ae) {
        if (ae.getActionCommand().equals("run")) {

            /* JOptionPane.showMessageDialog(null,
             "Run button is not implemented", "Information",
             JOptionPane.INFORMATION_MESSAGE);
             */

            int selfHdrId = 0;
            int clusterHdrId = 0;
            String approach = "S";
            int alphabetSize = 2;
```

```
int dimension = 8;
int variationType = 0;
int matchingRuleId = 0;

if (comDataSource.getSelectedItem() == "Primes 1") {

    selfHdrId = 1;
    clusterHdrId = 1;

} else if (comDataSource.getSelectedItem() == "Primes 2") {

    selfHdrId = 1;
    clusterHdrId = 2;

} else if (comDataSource.getSelectedItem() == "Primes 3") {

    selfHdrId = 1;
    clusterHdrId = 3;

} else if (comDataSource.getSelectedItem() == "Primes 4") {

    selfHdrId = 1;
    clusterHdrId = 4;

} else if (comDataSource.getSelectedItem() == "IES 1") {

    selfHdrId = 2;
    clusterHdrId = 5;

} else if (comDataSource.getSelectedItem() == "IES 2") {

    selfHdrId = 2;
    clusterHdrId = 6;

} else if (comDataSource.getSelectedItem() == "IES 3") {

    selfHdrId = 2;
    clusterHdrId = 7;

} else if (comDataSource.getSelectedItem() == "IES 4") {

    selfHdrId = 2;
    clusterHdrId = 8;

} else if (comDataSource.getSelectedItem() == "IES C") {

    selfHdrId = 2;
    clusterHdrId = 13;

}

if (comMatchingRule.getSelectedItem() == "R-contiguous") {

    matchingRuleId = 1;

} else if (comMatchingRule.getSelectedItem()
        == "Hamming Distance") {

    matchingRuleId = 2;
}

if (comVariation.getSelectedItem() == "Mutation") {

    variationType = 0;

} else if (comVariation.getSelectedItem()
        == "Recombination & Mutation") {

    variationType = 1;
}
```

```
Algorithm.setDBConnection(CustomConnection.getDBConnection__());

/*

 System.out.println(
 "Connection " + CustomConnection.getDBConnection__());

 System.out.println("selfHdrId " + selfHdrId);

 System.out.println("clusterHdrId " + clusterHdrId);

 System.out.println("approach " + approach);

 System.out.println(
 "Repertoire Size "
 + Integer.parseInt(tfRepertoireSize.getText()));

 System.out.println(
 "Prob Failure "
 + Double.parseDouble(
 tfProbOfNotMatchingSelf.getText()));

 System.out.println(
 "Matching Threshold "
 + Integer.parseInt(
 comMatchingThreshold.getSelectedItem().toString()));

 System.out.println(
 "Matching Rule "
 + comMatchingRule.getSelectedItem().toString().substring(
 0, 1));

 System.out.println("Variation " + variationType);

 System.out.println(
 "Number of Trials "
 + Integer.parseInt(tfNumberOfTrials.getText()));

 System.out.println("Alphabet Size " + alphabetSize);

 */

/*
 Algorithm algorithmRecord = new Algorithm(selfHdrId,
 clusterHdrId, approach,
 Integer.parseInt(tfRepertoireSize.getText()),
 Double.parseDouble(tfProbOfNotMatchingSelf.getText()),
 Integer.parseInt(
 comMatchingThreshold.getSelectedItem().toString()),
 matchingRuleId,
 variationType,
 Integer.parseInt(tfNumberOfTrials.getText()),
 alphabetSize);
 */

/* Algorithm algorithmRecord =
 new Algorithm(2, 6, "S", 0, 0.1, 8, 1, 0, 100, 2);*/


Algorithm algorithmRecord = new Algorithm(selfHdrId,
        clusterHdrId, approach,
        Integer.parseInt(tfRepertoireSize.getText()),
        Double.parseDouble(tfProbOfNotMatchingSelf.getText()),
        Integer.parseInt(
        comMatchingThreshold.getSelectedItem().toString()),
        matchingRuleId,
        variationType,
        Integer.parseInt(tfNumberOfTrials.getText()),
        alphabetSize);

/* Begin of statements to run the Algorithm */
```

```
                Scenario scenarioRecord = algorithmRecord.setScenarioRecord();

                Self[] self = algorithmRecord.setSelfArray();

                ClusterAssorted[] clusterAssortedArray =
algorithmRecord.setClusterAssortedArray();

                Detector[] competentDetectors =
algorithmRecord.generateInitialPopulation(
                        self, clusterAssortedArray);

                algorithmRecord.evolvePopulation(self, clusterAssortedArray,
                        competentDetectors);

                scenarioRecord.setStatistics();

                /* End of statements to run the Algorithm */

            } else {
                if (ae.getActionCommand().equals("cancel")) {

                    JOptionPane.showMessageDialog(null,
                            "The screen will be closed", "Information",
                            JOptionPane.INFORMATION_MESSAGE);

                    hide();

                } else {
                    if (ae.getActionCommand().equals("help")) {
                        JOptionPane.showMessageDialog(null,
                                "SANSAD: Screen to enter parameters",
                                "Information", JOptionPane.INFORMATION_MESSAGE);
                    }
                }
            }
        }
    }
}


class AlgorithmFrameLayout implements LayoutManager {

    public AlgorithmFrameLayout() {}

    public void addLayoutComponent(String name, Component comp) {}

    public void removeLayoutComponent(Component comp) {}

    public Dimension preferredLayoutSize(Container parent) {
        Dimension dim = new Dimension(0, 0);

        Insets insets = parent.getInsets();

        dim.width = 454 + insets.left + insets.right;
        dim.height = 329 + insets.top + insets.bottom;

        return dim;
    }

    public Dimension minimumLayoutSize(Container parent) {
        Dimension dim = new Dimension(0, 0);

        return dim;
    }

    public void layoutContainer(Container parent) {
        Insets insets = parent.getInsets();

        Component c;
```

```
c = parent.getComponent(0);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 24, 120, 24);
}
c = parent.getComponent(1);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 24, 128, 24);
}
c = parent.getComponent(2);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 88, 152, 24);
}
c = parent.getComponent(3);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 88, 72, 24);
}
c = parent.getComponent(4);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 112, 152, 24);
}
c = parent.getComponent(5);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 112, 72, 24);
}
c = parent.getComponent(6);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 136, 152, 24);
}
c = parent.getComponent(7);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 136, 72, 24);
}
c = parent.getComponent(8);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 160, 152, 24);
}
c = parent.getComponent(9);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 160, 176, 24);
}
c = parent.getComponent(10);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 184, 152, 24);
}
c = parent.getComponent(11);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 184, 176, 24);
}
c = parent.getComponent(12);
if (c.isVisible()) {
    c.setBounds(insets.left + 48, insets.top + 208, 152, 24);
}
c = parent.getComponent(13);
if (c.isVisible()) {
    c.setBounds(insets.left + 216, insets.top + 208, 72, 24);
}
c = parent.getComponent(14);
if (c.isVisible()) {
    c.setBounds(insets.left + 96, insets.top + 272, 88, 24);
}
c = parent.getComponent(15);
if (c.isVisible()) {
    c.setBounds(insets.left + 184, insets.top + 272, 88, 24);
}
c = parent.getComponent(16);
if (c.isVisible()) {
    c.setBounds(insets.left + 272, insets.top + 272, 88, 24);
}
c = parent.getComponent(17);
if (c.isVisible()) {
    c.setBounds(insets.left + 8, insets.top + 8, 440, 56);
```

```
        }
        c = parent.getComponent(18);
        if (c.isVisible()) {
            c.setBounds(insets.left + 8, insets.top + 72, 440, 168);
        }
    }
}
```

Appendix H – Java Source Code for ConnectionDialog.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.sql.*;


public class ConnectionDialog extends JInternalFrame {

    JLabel labelSourceUrl;
    JTextField textfieldSourceUrl;
    JLabel labelUser;
    JTextField textfieldUser;
    JLabel labelPassword;
    JPasswordField passwordField;
    JButton buttonConnect;
    JButton buttonCancel;
    JPanel panelConnection;

    public ConnectionDialog() {
        super("Connection Dialog", true, true, true, true);
        ConnectionDialogLayout customLayout = new ConnectionDialogLayout();

        getContentPane().setFont(new Font("Helvetica", Font.PLAIN, 12));
        getContentPane().setLayout(customLayout);
        labelSourceUrl = new JLabel("Source URL:");
        getContentPane().add(labelSourceUrl);
        textfieldSourceUrl = new JTextField("jdbc:odbc:vaio");
        textfieldSourceUrl.setBorder(new BevelBorder(BevelBorder.LOWERED));
        getContentPane().add(textfieldSourceUrl);
        labelUser = new JLabel("User:");
        getContentPane().add(labelUser);
        textfieldUser = new JTextField("ljgonzal");
        textfieldUser.setBorder(new BevelBorder(BevelBorder.LOWERED));
        getContentPane().add(textfieldUser);
        labelPassword = new JLabel("Password:");
        getContentPane().add(labelPassword);
        passwordField = new JPasswordField();
        passwordField.setBorder(new BevelBorder(BevelBorder.LOWERED));
        getContentPane().add(passwordField);
        buttonConnect = new JButton("Connect");
        buttonConnect.setBorder(new BevelBorder(BevelBorder.RAISED));
        buttonConnect.setActionCommand("connect");

        // buttonConnect.addActionListener(this);
        buttonConnect.addActionListener(new PushButtonActionListener());
        getContentPane().add(buttonConnect);
        buttonCancel = new JButton("Quit");
        buttonCancel.setBorder(new BevelBorder(BevelBorder.RAISED));
        buttonCancel.setActionCommand("quit");

        // buttonCancel.addActionListener(this);
        buttonCancel.addActionListener(new PushButtonActionListener());
        getContentPane().add(buttonCancel);
        panelConnection = new JPanel();
        panelConnection.setBorder(
                new TitledBorder(new LineBorder(Color.black, 1),
                "Connection Parameters"));
        getContentPane().add(panelConnection);
        setSize(getPreferredSize());

        /*
         addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
         System.exit(0);
         }
         });
```

```
         */
    }

    /*
     public static void main(String args[]) {
     ConnectionDialog window = new ConnectionDialog();

     window.setTitle("ConnectionDialog");
     window.pack();
     window.show();
     }
     */
    private class PushButtonActionListener implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
            if (ae.getActionCommand().equals("connect")) {

                /* JOptionPane.showMessageDialog(null,
                 "Connect button is not implemented", "Information",
                 JOptionPane.INFORMATION_MESSAGE);
                 */

                /** @pre $none */


                char[] passwordIn = passwordField.getPassword();

                /* CustomConnection customConnectionRecord = new CustomConnection(
                 textfieldSourceUrl.getText().toString(),
                 textfieldUser.getText().toString(), passwordIn);

                 Connection dBConnection = customConnectionRecord.getDBConnection();

                 */

                CustomConnection.setDBConnection__(
                        textfieldSourceUrl.getText().toString(),
                        textfieldUser.getText().toString(), passwordIn);

                Connection dBConnection = CustomConnection.getDBConnection__();

                if (dBConnection != null) {
                    JOptionPane.showMessageDialog(null,
                            "Connection has been established", "Information",
                            JOptionPane.INFORMATION_MESSAGE);
                    hide(); // Close Connection Dialog

                } else {

                    JOptionPane.showMessageDialog(null,
                            "Connection has not been established", "Information",
                            JOptionPane.INFORMATION_MESSAGE);

                }

            } else {
                if (ae.getActionCommand().equals("quit")) {

                    /*
                     JOptionPane.showMessageDialog(null,
                     "Cancel button is not implemented", "Information",
                     JOptionPane.INFORMATION_MESSAGE);
                     */
                    System.exit(0);
                }
            }
        }
    }
}


class ConnectionDialogLayout implements LayoutManager {
```

```
    public ConnectionDialogLayout() {}

    public void addLayoutComponent(String name, Component comp) {}

    public void removeLayoutComponent(Component comp) {}

    public Dimension preferredLayoutSize(Container parent) {
        Dimension dim = new Dimension(0, 0);
        Insets insets = parent.getInsets();

        dim.width = 327 + insets.left + insets.right;
        dim.height = 241 + insets.top + insets.bottom;
        return dim;
    }

    public Dimension minimumLayoutSize(Container parent) {
        Dimension dim = new Dimension(0, 0);

        return dim;
    }

    public void layoutContainer(Container parent) {
        Insets insets = parent.getInsets();
        Component c;

        c = parent.getComponent(0);
        if (c.isVisible()) {
            c.setBounds(insets.left + 64, insets.top + 40, 72, 24);
        }
        c = parent.getComponent(1);
        if (c.isVisible()) {
            c.setBounds(insets.left + 144, insets.top + 40, 104, 24);
        }
        c = parent.getComponent(2);
        if (c.isVisible()) {
            c.setBounds(insets.left + 64, insets.top + 64, 72, 24);
        }
        c = parent.getComponent(3);
        if (c.isVisible()) {
            c.setBounds(insets.left + 144, insets.top + 64, 104, 24);
        }
        c = parent.getComponent(4);
        if (c.isVisible()) {
            c.setBounds(insets.left + 64, insets.top + 88, 72, 24);
        }
        c = parent.getComponent(5);
        if (c.isVisible()) {
            c.setBounds(insets.left + 144, insets.top + 88, 104, 24);
        }
        c = parent.getComponent(6);
        if (c.isVisible()) {
            c.setBounds(insets.left + 64, insets.top + 168, 88, 24);
        }
        c = parent.getComponent(7);
        if (c.isVisible()) {
            c.setBounds(insets.left + 152, insets.top + 168, 88, 24);
        }
        c = parent.getComponent(8);
        if (c.isVisible()) {
            c.setBounds(insets.left + 8, insets.top + 8, 312, 144);
        }
    }
}
```

Appendix I – Java Source Code for Algorithm.java

```
/*
 * @(#)Algorithm.java 1.00 08/20/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.text.DecimalFormat;
import java.util.Random;
import javax.swing.JOptionPane;


/**
 * Algorithm class
 * @author Luis Gonzalez
 * @version 1.4.2
 */

public class Algorithm {

    /*
     *  Algorithm Class - SANSAD Algorithm
     *  Generate and evolve initial population of detectors
     */

    /**
     * @seen Base to calculate max integer
     */
    private static final int BASE = 2;

    // Class variable

    /*
     * Note on dimension variable: If you have an 8-bit vector,
     * then 8 is the dimension. If you have a single integer,
     * then 1 is the dimension
     *
     */

    private static Connection dBConnection = null; // Connection
    private static int maxInteger = 0;
    private static int range = 0;

    // Instance variables

    private int selfHdrId;
    private int clusterHdrId;
    private String approach;
    private int repertoireSize = 0;
    private double probOfNotMatchingSelf;
    private int matchingThreshold;
    private int matchingRule;
    private int variationType;
    private int numberOfTrials;
    private int alphabetSize = 0;
    private int stringLength = 0;
    private int dimension = 0;
    private int scenarioId = 0;
    private int initialPopulation = 0;
    private int totalNonSelfRowsInTest = 0;
    private int totalRowsSelfData = 0;
    private int totalRowsTestData = 0;
```

```
// Constructor

public Algorithm() {
    selfHdrId = 0;
    clusterHdrId = 0;
    approach = null;
    repertoireSize = 0;
    probOfNotMatchingSelf = 0;
    matchingThreshold = 0;
    matchingRule = 0;
    variationType = 0;
    numberOfTrials = 0;
    alphabetSize = 0;
    stringLength = 0;
    dimension = 0;
    scenarioId = 0;
    initialPopulation = 0;
    totalRowsSelfData = 0;
    totalRowsTestData = 0;
    totalNonSelfRowsInTest = 0;
}

public Algorithm(
        int selfHdrIdIn,
        int clusterHdrIdIn,
        String approachIn,
        int repertoireSizeIn,
        double probOfNotMatchingSelfIn,
        int matchingThresholdIn,
        int matchingRuleIn,
        int variationTypeIn,
        int numberOfTrialsIn,
        int alphabetSizeIn) {

    selfHdrId = selfHdrIdIn;
    clusterHdrId = clusterHdrIdIn;
    approach = approachIn;
    repertoireSize = repertoireSizeIn;
    probOfNotMatchingSelf = probOfNotMatchingSelfIn;
    matchingThreshold = matchingThresholdIn;
    matchingRule = matchingRuleIn;
    variationType = variationTypeIn;
    numberOfTrials = numberOfTrialsIn;
    alphabetSize = alphabetSizeIn;
    scenarioId = 0;
    initialPopulation = 0;
    totalRowsSelfData = 0;
    totalRowsTestData = 0;
    totalNonSelfRowsInTest = 0;
}

/**
 * Method to create new Scenario
 *
 * @return scenarioRecord
 */

public Scenario setScenarioRecord() {

    // Scenario
    Scenario scenarioRecord = new Scenario(this.selfHdrId, this.clusterHdrId,
            this.repertoireSize, this.probOfNotMatchingSelf,
            this.matchingThreshold, this.matchingRule, this.variationType,
            this.numberOfTrials, this.alphabetSize, this.approach);

    scenarioRecord.setDBConnection(dBConnection);

    scenarioRecord.createScenario();

    scenarioId = scenarioRecord.getScenarioId();
    initialPopulation = scenarioRecord.getInitialPopulation();
```

```
    repertoireSize = scenarioRecord.getRepertoireSize();
    totalRowsSelfData = scenarioRecord.getTotalRowsSelfData();
    totalRowsTestData = scenarioRecord.getTotalRowsTestData();
    totalNonSelfRowsInTest = scenarioRecord.getTotalNonSelfRowsInTest();
    stringLength = scenarioRecord.getStringLength();

    this.dimension = stringLength;

    return scenarioRecord;

}

/**
 * Method to set SELF Array
 *
 * @return self
 */

public Self[] setSelfArray() {

    Self[] self = Self.setSelfArray(dBConnection, this.selfHdrId,
            this.totalRowsSelfData);

    for (int x = 0; x < self.length; x++) {
        self[x].getRealValue();
        self[x].getBinaryString();
    }

    return self;

}

/**
 * Method to set CLUSTER_ASSORTED Array
 *
 * @return clusterAssortedArray
 */

public ClusterAssorted[] setClusterAssortedArray() {

    ClusterAssorted[] clusterAssortedArray = ClusterAssorted.setClusterAssortedArray(
            dBConnection, this.selfHdrId, this.clusterHdrId,
            this.totalRowsTestData);

    for (int x = 0; x < clusterAssortedArray.length; x++) {
        clusterAssortedArray[x].getRealValue();
        clusterAssortedArray[x].getBinaryString();
    }

    return clusterAssortedArray;

}

/**
 * Method to set DETECTORS Array
 *
 * @return competent detectors
 */

public Detector[] generateInitialPopulation(
        final Self[] selfIn,
        final ClusterAssorted[] clusterAssortedArrayIn) {

    Self[] self = selfIn;
    ClusterAssorted[] clusterAssortedArray = clusterAssortedArrayIn;
    Detector[] competentDetectors = new Detector[this.initialPopulation];

    DecimalFormat df = new DecimalFormat("0.00");
    Random rand = new Random();
    String candidateDetectorBinary = new String();
```

```
boolean flagDuplicatedDetectors = false;
double mutationStepSize = 0;
double mutationStepSizeMutated = 0;
int affinityMean = 0;
int candidateDetectorInteger = 0;
int trialId = 1;
long begin = 0;
long end = 0;
long trialDuration = 0;

System.out.println();
System.out.println("Generating Initial Population.  Please wait ...\n");

// Start of the cycle to generate detectors

// For initial population the process is:
// (1) enter detector;
// (2) check duplicity;
// (3) calculate affinity;
// (4) add to repertoire

// Set max integer

maxInteger = (int) Math.pow(BASE, this.stringLength) - 1;

// Set range

range = (int) Math.pow(BASE, this.stringLength);

begin = System.currentTimeMillis();

for (int x = 0; x < this.initialPopulation; x++) {

    int counterDuplicity = 0;

    // Do Begin to accept or reject a detector based on affinity mean

    do {

        do {
            // Generate random number

            candidateDetectorInteger = Math.abs(rand.nextInt(range));

            if (x > 0) {
                flagDuplicatedDetectors = Detector.findDuplicatedDetectors(
                        candidateDetectorInteger, competentDetectors, x);
            }

            counterDuplicity++;
        } while (flagDuplicatedDetectors == true);

        // Calculate Affinity Mean
        affinityMean = Detector.calculateAffinityDetectorSelf(
                candidateDetectorInteger, self, this.matchingRule,
                this.matchingThreshold);

    } while (affinityMean >= this.matchingThreshold);

    // Do End to accept or reject a detector based on affinity mean

    // Convert Integer to Binary String
    candidateDetectorBinary = Integer.toBinaryString(
            candidateDetectorInteger);

    if (candidateDetectorBinary.length() < this.stringLength) {
        candidateDetectorBinary = StringFunctions.lPad(
                candidateDetectorBinary, this.stringLength, "0");

    }
```

```
            mutationStepSize = Detector.calculateStdev(candidateDetectorInteger);

            competentDetectors[x] = new Detector(x + 1, this.selfHdrId,
                    this.clusterHdrId, this.scenarioId, trialId,
                    candidateDetectorInteger, affinityMean,
                    candidateDetectorBinary,
                    Double.parseDouble(df.format(mutationStepSize)));

            Detector.setDBConnection(dBConnection);

        }

        // End of the cycle to generate detectors

        end = System.currentTimeMillis();

        trialDuration = end - begin;

        // Trial
        Trial trialRecord = new Trial(trialId, this.scenarioId, this.selfHdrId,
                this.clusterHdrId, this.matchingRule);

        trialRecord.setDBConnection(dBConnection);

        // Insert Trial Record
        trialRecord.insertTrialRow();

        // Insert Trial for initial population
        System.out.print("Trial " + trialId + ": ");

        // Insert Detector Record
        for (int x = 0; x < this.initialPopulation; x++) {
            competentDetectors[x].insertDetectorRow();
        }

        // Update trialDuration
        trialRecord.setTrialDuration(trialDuration);
        trialRecord.setTotalDetection(this.totalRowsTestData,
                this.repertoireSize, this.matchingThreshold,
                clusterAssortedArray, competentDetectors);
        trialRecord.setTotalTruePositive();
        trialRecord.setTotalTrueNegative();
        trialRecord.setTotalFalsePositive();
        trialRecord.setTotalFalseNegative();
        trialRecord.updateTrialRow();

        // Generate statistics
        trialRecord.displayStatistics();

        return competentDetectors;

    }

    /**
     * Evolve Population
     *
     * @param selfIn
     * @param clusterAssortedArrayIn
     * @param competentDetectorsIn
     */

    public void evolvePopulation(
            Self[] selfIn,
            ClusterAssorted[] clusterAssortedArrayIn,
            Detector[] competentDetectorsIn) {

        Self[] self = selfIn;
        ClusterAssorted[] clusterAssortedArray = clusterAssortedArrayIn;
        Detector[] competentDetectors = competentDetectorsIn;

        DecimalFormat df = new DecimalFormat("0.00");
```

```
Random rand = new Random();
String candidateDetectorBinary = new String();
String offSpringMutatedBinary = new String();
String parent1 = null;
String parent2 = null;
boolean flagDuplicatedDetectors = false;
double mutationStepSize = 0;
double mutationStepSizeMutated = 0;
int affinityMean = 0;
int bestParentIndex = 0;
int candidateDetectorInteger = 0;
int offSpringMutated = 0;
int parent1Index = 0;
int parent2Index = 0;
int preParent1Index = 0;
int preParent2Index = 0;
int trialId = 1;
long begin = 0;
long end = 0;
long trialDuration = 0;

// Evolve Initial Repertoire

System.out.println();
System.out.println("Evolving population...\n");

mutationStepSize = 0;
do {

    // Trial

    trialId++;

    System.out.print("Trial " + trialId + ": ");

    Trial trialRecord = new Trial(trialId, this.scenarioId,
            this.selfHdrId, this.clusterHdrId, this.matchingRule);

    // Insert Trial Record

    trialRecord.insertTrialRow();

    int x = 0;

    // Start of the cycle to generate detectors
    begin = System.currentTimeMillis();

    while (x < this.repertoireSize) {

        // Do Begin to accept or reject detector based on affinity mean


        do {

            parent1Index = rand.nextInt(this.initialPopulation);

            parent2Index = rand.nextInt(this.initialPopulation);

            parent1 = competentDetectors[parent1Index].getBinaryString();

            parent2 = competentDetectors[parent2Index].getBinaryString();

            // Select the best parent

            if (competentDetectors[parent1Index].getAffinity()
                    > competentDetectors[parent2Index].getAffinity()) {
                bestParentIndex = parent2Index;
            } else {
                bestParentIndex = parent1Index;
            }
```

```
        candidateDetectorInteger = competentDetectors[bestParentIndex].getRealValue();

         mutationStepSize = competentDetectors[bestParentIndex].getMutationStepSize();

                        // 1 =  Recombination and Mutation; otherwise only mutation

                        if (this.variationType == 1) {

                            // Recombination (Crossover)

                            candidateDetectorInteger = doSinglePointCrossover(
                                    parent1, parent2);

                            mutationStepSize = Detector.calculateStdev(
                                    candidateDetectorInteger);

                        }

                        // Mutation

                        mutationStepSizeMutated = Double.parseDouble(
                                df.format(
                                        Detector.mutateMutationStepSize(
                                                this.dimension, mutationStepSize)));

                        offSpringMutated = (int) Math.round(
                                Detector.mutateGene(candidateDetectorInteger,
                                mutationStepSizeMutated));

                        // Convert offSpringMutated to binary and
                        // trunc offSpringMutatedBinary.length to stringLength

                        offSpringMutatedBinary = Integer.toBinaryString(
                                offSpringMutated);

                        if (offSpringMutatedBinary.length() != this.stringLength) {
                            offSpringMutatedBinary = StringFunctions.lPad(
                                    offSpringMutatedBinary, this.stringLength, "0");
                        }

                        // Gen Repair

                        if (offSpringMutated > maxInteger) {
                            offSpringMutated = StringFunctions.toDecimal(
                                    offSpringMutatedBinary);
                        }

                        // Calculate Affinity Mean
                        affinityMean = Detector.calculateAffinityDetectorSelf(
                                offSpringMutated, self, this.matchingRule,
                                this.matchingThreshold);

                    } while (affinityMean >= this.matchingThreshold);

                    // Tournament selection

                    if (competentDetectors[bestParentIndex].getAffinity()
                            < affinityMean) {

                        offSpringMutatedBinary =
    competentDetectors[bestParentIndex].getBinaryString();
                        offSpringMutated =
    competentDetectors[bestParentIndex].getRealValue();
                        mutationStepSizeMutated = Double.parseDouble(
                                df.format(

    competentDetectors[bestParentIndex].getMutationStepSize()));
                        affinityMean = competentDetectors[bestParentIndex].getAffinity();
                    }

                    // End tournament selection
```

```
            // Do End DetectorEvaluation


            flagDuplicatedDetectors = Detector.findDuplicatedDetectors(
                    offSpringMutated, competentDetectors, x);

            if (flagDuplicatedDetectors == false) {

                competentDetectors[x] = new Detector(x + 1, this.selfHdrId,
                        this.clusterHdrId, this.scenarioId, trialId,
                        offSpringMutated, affinityMean,
                        offSpringMutatedBinary, mutationStepSizeMutated);

                x++;
            }
        }

        // End of the cycle to generate detectors

        end = System.currentTimeMillis();

        trialDuration = end - begin;

        // Update trialDuration
        trialRecord.setTrialDuration(trialDuration);
        trialRecord.setTotalDetection(this.totalRowsTestData,
                this.repertoireSize, this.matchingThreshold,
                clusterAssortedArray, competentDetectors);
        trialRecord.setTotalTruePositive();
        trialRecord.setTotalTrueNegative();
        trialRecord.setTotalFalsePositive();
        trialRecord.updateTrialRow();

        // Insert Detector Record

        x = 0;

        while (x < this.repertoireSize) {
            competentDetectors[x].insertDetectorRow();
            x++;
        }

        // Display statistics

        trialRecord.displayStatistics();

    } while (trialId < this.numberOfTrials);

}

/**
 * Set Connection
 *
 * @param dbConnection
 * @return Connection
 */

public static Connection setDBConnection(final Connection dBConnectionIn) {
    dBConnection = dBConnectionIn;
    return dBConnection;
}

/**
 * Set Connection
 *
 * @param sourceURLIn
 * @param userIn
 * @param passwordIn
 * @return dBConnection
 */
```

```
    public static Connection setDBConnection(final String sourceURLIn,
            final String userIn,
            final String passwordIn) {
        try {
            String driver = "sun.jdbc.odbc.JdbcOdbcDriver";

            // Load the driver class
            Class.forName(driver);
            dBConnection = DriverManager.getConnection(sourceURLIn, userIn,
                    passwordIn);
        } catch (ClassNotFoundException cnfe) {
            System.err.println(cnfe);
        } catch (SQLException sqle) {
            System.err.println(sqle);
        }

        return dBConnection;
    }

    /**
     * Do a single point crossover
     *
     * @param parent1
     * @param parent2
     * @return offSpringInteger
     */
    private static int doSinglePointCrossover(String parent1, String parent2) {
        StringBuffer offSpring = new StringBuffer();
        Random rand = new Random();
        int stringLength = parent1.length();
        int singlePoint = 0;
        int offSpringInteger = 0;

        do {
            while (singlePoint == 0) {
                singlePoint = rand.nextInt(stringLength);
            }
            offSpring = offSpring.append(parent1.substring(0, singlePoint)).append(
                    parent2.substring(singlePoint, stringLength));
            offSpringInteger = StringFunctions.toDecimal(offSpring.toString());
        }while (offSpringInteger == 0);
        return offSpringInteger;
    }

    /*
     * SANSAD Algorithm - Main Body
     *
     * @throws java.sql.SQLException
     */
    public static void main(String[] args) throws java.sql.SQLException {
        try {
            JOptionPane.showConfirmDialog(null,
                    "SANSAD will be started.\n" + "Do you want to continue?",
                    "Warning", JOptionPane.YES_NO_OPTION);

            setDBConnection("jdbc:odbc:vaio", "ljgonzal", "ljgonzal");

            // Only Mutation

            Algorithm algorithmRecord = new Algorithm(1, 4, "S", 0, 0, 5, 1, 0,
                    20, 2);

            Scenario scenarioRecord = algorithmRecord.setScenarioRecord();

            Self[] self = algorithmRecord.setSelfArray();

            ClusterAssorted[] clusterAssortedArray =
algorithmRecord.setClusterAssortedArray();

            Detector[] competentDetectors = algorithmRecord.generateInitialPopulation(
```

```
                    self, clusterAssortedArray);

            algorithmRecord.evolvePopulation(self, clusterAssortedArray,
                    competentDetectors);

            scenarioRecord.setStatistics();
        }
        finally {
            if (dBConnection != null && dBConnection.isClosed()) {
                dBConnection.close();
            }
        }
    }
}
```

Appendix J – Java Source Code for MatchingRule.java

```
/*
 * @(#)MatchingRule.java 1.00 08/25/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */

/** MatchingRule class
 */

/**
 * @author  Luis Gonzalez
 * @version 1.4.2
 */

public class MatchingRule {

    // Instance variables

    private int matchingRuleId;
    private String binaryString;
    private int distance;

    /** Constructor to initialize class.
     */

    public MatchingRule() {
        matchingRuleId = 0;
        binaryString = null;
        distance = 0;

    }

    /** Constructor to initialize class.
     */
    public MatchingRule(
            final int matchingRuleIdIn,
            final String binaryStringIn) {

        matchingRuleId = matchingRuleIdIn;
        binaryString = binaryStringIn;
        distance = 0;

    }

    /** Get distance
     */

    /**
     * @return distance
     */

    public final int getDistance() {

        if (this.matchingRuleId == 1) {

            distance = this.getRContiguousDistance();

        } else if (this.matchingRuleId == 2) {

            distance = this.getHammingDistance();
        }

        return distance;
    }
```

```
/** Calculate Hamming Distance
 */

/**
 * @return HammingDistance
 */

private int getHammingDistance() {
    int hammingDistance = 0;
    int textLength = this.binaryString.length();

    for (int x = 0; (x < textLength); x++) {

        // Check for 1
        char ch = this.binaryString.charAt(x);

        if (Character.isDigit(this.binaryString.charAt(x)) == true) {
            if (ch == '1') {
                hammingDistance = hammingDistance + 1;
            }
        } else {
            hammingDistance = -1;
            break;
        }
    }

    return hammingDistance;
}

/** Calculate R-Contiguous Distance
 */

/**
 * @return R-Contiguous Distance
 */
private int getRContiguousDistance() {
    int indexOneBegin = -1;
    int indexOneEnd = -1;
    int lastIndexOneEnd = this.binaryString.lastIndexOf('1');
    int lastIndexZeroEnd = this.binaryString.lastIndexOf('0');
    int exitFlag = 0;
    int length0 = 0;
    int length1 = 0;

    while (exitFlag != -1) {
        indexOneBegin = this.binaryString.indexOf('1', indexOneEnd);
        indexOneEnd = this.binaryString.indexOf('0', indexOneEnd + 1);
        exitFlag = indexOneEnd;
        if (indexOneBegin == -1) {
            indexOneBegin = lastIndexZeroEnd + 1;
        }
        if (indexOneEnd == -1) {
            indexOneEnd = lastIndexOneEnd + 1;
        }
        length1 = indexOneEnd - indexOneBegin;
        if (length0 < length1) {
            length0 = length1;
        }
        length1 = 0;
    }

    return length0;
}

/* Method to obtain XOR given two strings
 */

/*
 static StringBuffer getXOR(String binaryStringIn, String string2In) {
 int result = 0;
```

```
    StringBuffer XOROutput = new StringBuffer();

    for (int x = 0; (binaryStringIn.length() == string2In.length())
    & (x < binaryStringIn.length()); x++) {
    result = binaryStringIn.substring(x, x + 1).compareTo(
    string2In.substring(x, x + 1));
    if (result == 0) {
    XOROutput = XOROutput.append("0");
    } else {
    XOROutput = XOROutput.append("1");
    }
    }
    return XOROutput;
    }
    */



  /* Method to determine if a given string has n (matchingThreshold) contiguous bits
    */

    /*
    static boolean getRContiguousMatch(String binaryStringIn, int matchingThreshold) {
    boolean RContiguousFlag = false;
    int acum = 0;
    int result = 0;

    for (int x = 0; ((binaryStringIn.length() >= matchingThreshold)
    & (x <= binaryStringIn.length() - 2)
    & (acum != matchingThreshold - 1)); x++) {
    result = binaryStringIn.substring(x, x + 1).compareTo(
    binaryStringIn.substring(x + 1, x + 2));
    if (result == 0) {
    acum = acum + 1;
    } else {
    acum = 0;
    }
    }
    if (acum == (matchingThreshold - 1)) {
    RContiguousFlag = true;
    }
    return RContiguousFlag;
    }
    */

   public static void main(String[] args) {

       String xorOutput = "1011111000111";

       MatchingRule matchingRuleRecord = new MatchingRule(1, xorOutput);

       System.out.println(
               "RContiguous Distance is " + matchingRuleRecord.getDistance());

       matchingRuleRecord = new MatchingRule(2, xorOutput);

       System.out.println(
               "Hamming Distance is " + matchingRuleRecord.getDistance());

   }

}
```

Appendix K – Java Source Code for SelfHeader.java

```
/*
 * @(#)SelfHeader.java 1.00 09/05/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


/** SelfHeader
 */

/**
 * @author     Luis Gonzalez
 * @version    1.4.2
 */
public class SelfHeader {

    // Class variables
    private static Connection dBConnection__ = null;
    private static PreparedStatement theStatement__ = null;
    private static ResultSet theResultSet__ = null;

    // Instance variables
    private int selfHdrId;
    private int binaryStringLength;
    private String description;

    /** Constructor
     */

    public SelfHeader() {
        selfHdrId = 0;
        binaryStringLength = 0;
        description = "";
    }

    /** Constructor
     */

    public SelfHeader(
            int binaryStringLengthIn,
            String descriptionIn) {

        selfHdrId = 0;
        binaryStringLength = binaryStringLengthIn;
        description = descriptionIn;
    }

    /** Set dBConnection
     */

    /**
     * @param dBConnectionIn
     */

    public static void setDBConnection(Connection dBConnectionIn) {
        dBConnection__ = dBConnectionIn;

    }
```

```
/** Create SelfHeader
 */

public void createSelfHeader() {

    this.setNextSelfHdrId();
    this.insertSelfHeaderRow();

}

/** Set NextSelfHdrId
 */

private final void setNextSelfHdrId() {
    try {
        theStatement__ = dBConnection__.prepareStatement(
                "SELECT (nvl(max(id),0)+1) FROM self_header");

        theResultSet__ = theStatement__.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet__.next()) {
            selfHdrId = theResultSet__.getInt(1);
        }

        theResultSet__.close();
        theStatement__.close();

    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/** Get SelfHdrId
 */

/**
 * @return selfHdrId
 */

public int getSelfHdrId() {
    return selfHdrId;
}

/** Get SelfBinaryStringLength
 */

/**
 * @return binaryStringLength
 */

public int getBinaryStringLength() {
    return binaryStringLength;
}

/** Get Description
 */

/**
 * @return description
 */

public String getDescription() {
    return description;
}

/** Insert Self Header Row
 */

private void insertSelfHeaderRow() {
    try {
```

```
            // Primary Key (id)

            theStatement__ = dBConnection__.prepareStatement(
                    "INSERT INTO SELF_HEADER (id, description,"
                            + " binary_string, creation_date, last_update_date)"
                            + " VALUES (?,?,?,?, sysdate, sysdate)");

            theStatement__.setInt(1, this.selfHdrId);
            theStatement__.setString(2, this.description);
            theStatement__.setInt(3, this.binaryStringLength);

            theStatement__.executeUpdate();

            theStatement__.close();

        } catch (SQLException sqle) {
            System.err.println(sqle);
            System.out.println("Error during Insert");
        }
    }

}
```

Appendix L – Java Source Code for Self.java

```
/*
 * @(#)Self.java 1.00 09/05/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


/** Self (Normal Activity) class
 */

/**
 * @author     Luis Gonzalez
 * @version    1.4.2
 */

public class Self extends SelfHeader {

    // Class variables
    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;
    private static ResultSet theResultSet = null;

    // Instance variables
    private int selfId;
    private int selfHdrId;
    private int realValue;
    private String binaryString;

    /** Constructor to initialize class.
     */

    public Self() {
        selfId = 0;
        selfHdrId = 0;
        realValue = 0;
        binaryString = null;
        return;
    }

    /** Constructor to initialize class.
     */

    public Self(
            final int selfIdIn,
            final int selfHdrIdIn,
            final int realValueIn,
            final String binaryStringIn) {

        selfId = selfIdIn;
        selfHdrId = selfHdrIdIn;
        realValue = realValueIn;
        binaryString = binaryStringIn;
        return;
    }

    /** Set DBConnection
     */

    /**
```

```
 * @param dBConnectionIn
 */


public static final void setDBConnection(final Connection dBConnectionIn) {
    dBConnection = dBConnectionIn;

}

/** Create Self.
 */


public final void createSelf() {

    this.insertSelfRow();

}

/** Get selfId
 */

/**
 * @return selfId
 */

public final int getSelfId() {
    return selfId;
}

/** Method to get selfHdrId
 */

/**
 * @return selfHdrId
 */


public final int getSelfHdrId() {
    return selfHdrId;
}

/** Get realValue
 */

/**
 * @return realValue
 */

public final int getRealValue() {
    return realValue;
}

/** Get binaryString
 */

/**
 * @return binaryString
 */

public final String getBinaryString() {
    return binaryString;
}

/** Set setSelfArray
 */

/**
 * @return self
 */
```

```
    public static Self[] setSelfArray(
            final Connection dBConnectionIn,
            final int selfHdrIdIn,
            final int totalRowsSelfDataIn) {

        // Populate SELF Array
        Self[] self = new Self[totalRowsSelfDataIn];

        dBConnection = dBConnectionIn;

        int rowIndex = 0;

        try {
            theStatement = dBConnection.prepareStatement(
                    "SELECT t1.id, t1.shdr_id, t1.real_value, "
                           + " t1.binary_string FROM self t1 WHERE t1.shdr_id = ?");

            theStatement.setInt(1, selfHdrIdIn);

            theResultSet = theStatement.executeQuery();

            // Fetch and Populate the cursor with SELF

            while (theResultSet.next()) {
                Self selfRecord = new Self(theResultSet.getInt(1),
                        theResultSet.getInt(2), theResultSet.getInt(3),
                        theResultSet.getString(4));

                self[rowIndex] = selfRecord;
                rowIndex++;
            }

            theResultSet.close();
            theStatement.close();

        } catch (SQLException sqle) {
            System.err.println(sqle);
        }
        return self;
    }

    /** Insert Self Row.
     */

    private void insertSelfRow() {
        try {

            // Primary Key (id, ch_shdr_id, ch_id)

            theStatement = dBConnection.prepareStatement(
                    "INSERT INTO SELF (id, shdr_id,"
                            + " binary_string, real_value, creation_date,"
                            + " last_update_date) VALUES (?,?,?,?, sysdate,"
                            + " sysdate)");

            theStatement.setInt(1, this.selfId);
            theStatement.setInt(2, this.selfHdrId);
            theStatement.setString(3, this.binaryString);
            theStatement.setInt(4, this.realValue);

            theStatement.executeUpdate();

            theStatement.close();

        } catch (SQLException sqle) {
            System.err.println(sqle);
            System.out.println("Error during Insert");
        }
    }

}
```

Appendix M – Java Source Code for ClusterHeader.java

```
/*
 * @(#)ClusterHeader.java 1.00 09/20/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * ClusterHeader class
 */

/**
 * @author  Luis Gonzalez
 * @version 1.4.2
 */
public class ClusterHeader {

    // Class variables
    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;
    private static ResultSet theResultSet = null;

    // Instance variables
    private int clusterHdrId;
    private int selfHdrId;
    private String description;

    /**
     * Constructor to initialize class.
     */
    public ClusterHeader() {
        clusterHdrId = 0;
        selfHdrId = 0;
        description = null;
    }

    /**
     * Constructor to initialize class
     *
     * @param clusterHdrIdIn
     * @param selfHdrIdIn
     */
    public ClusterHeader(final int clusterHdrIdIn, final int selfHdrIdIn) {
        clusterHdrId = clusterHdrIdIn;
        selfHdrId = selfHdrIdIn;
        description = null;
    }

    /** Method to set dBConnection
     */

    /**
     * @param dBConnectionIn
     */
    public static void setDBConnection(final Connection dBConnectionIn) {
        dBConnection = dBConnectionIn;
    }

    /** Create ClusterHeader.
     */
    public final void createClusterHeader() {
```

```
    this.setNextClusterHdrId();
    this.insertClusterHeaderRow();
}

/** Set NextclusterHdrId.
 */
private void setNextClusterHdrId() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT (nvl(max(id),0)+1)"
                        + " FROM cluster_header WHERE shdr_id = ?");
        theStatement.setInt(1, this.selfHdrId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            this.clusterHdrId = theResultSet.getInt(1);
        }
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/** Get clusterHdrId
 */

/**
 * @return clusterHdrId
 */
public final int getClusterHdrId() {
    return clusterHdrId;
}

/** Get SelfHdrId
 */

/**
 * @return selfHdrId
 */
public final int getSelfHdrId() {
    return selfHdrId;
}

/** Get Description
 */

/**
 * @return description
 */
public final String getDescription() {
    return description;
}

/** Insert cluster header row.
 */
private void insertClusterHeaderRow() {
    try {
        theStatement = dBConnection.prepareStatement(
                "INSERT INTO cluster_header (id, shdr_id, description,"
                        + " creation_date, last_update_date)"
                        + " VALUES (?,?,?, sysdate, sysdate)");
        theStatement.setInt(1, this.clusterHdrId);
        theStatement.setInt(2, this.selfHdrId);
        theStatement.setString(3, this.description);
        theStatement.executeUpdate();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
        System.out.println("Error during Insert");
    }
}}
```

Appendix N – Java Source Code for ClusterAssorted.java

```
/*
 * @(#)ClusterAssorted.java 1.00 09/20/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


/** ClusterAssorted class
 */

/**
 * @author     Luis Gonzalez
 * @version    1.4.2
 */
public class ClusterAssorted {

    // Class variables
    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;
    private static ResultSet theResultSet = null;

    // Instance variables
    private int clusterAssortedId;
    private int selfHdrId;
    private int clusterHdrId;
    private int realValue;
    private String binaryString;

    /** Constructor to initialize class.
     */
    public ClusterAssorted() {
        clusterAssortedId = 0;
        selfHdrId = 0;
        clusterHdrId = 0;
        realValue = 0;
        binaryString = "";
        return;
    }

    /** Constructor to initialize class.
     */
    public ClusterAssorted(final int clusterAssortedIdIn, final int selfHdrIdIn, final
int clusterHdrIdIn, final int realValueIn, final String binaryStringIn) {
        clusterAssortedId = clusterAssortedIdIn;
        selfHdrId = selfHdrIdIn;
        clusterHdrId = clusterHdrId;
        realValue = realValueIn;
        binaryString = binaryStringIn;
        return;
    }

    /** Set dBConnection
     */

    /**
     * @param dBConnectionIn
     */
    public static void setDBConnection(final Connection dBConnectionIn) {
        dBConnection = dBConnectionIn;
    }
```

```
    /** Create ClusterAssorted.
     */
    public final void createClusterAssorted() {
        this.insertClusterAssortedRow();
    }

    /** Set ClusterAssortedArray
     */

    /**
     * @return clusterAssortedArray
     */
    public static ClusterAssorted[] setClusterAssortedArray(final Connection
dBConnectionIn, final int selfHdrIdIn, final int clusterHdrIdIn, final int
totalRowsTestDataIn) {

        // Populate ClusterAssorted Array
        dBConnection = dBConnectionIn;
        int selfHdrId = selfHdrIdIn;
        int clusterHdrId = clusterHdrIdIn;
        int totalRowsTestData = totalRowsTestDataIn;
        ClusterAssorted[] clusterAssortedArray = new ClusterAssorted[totalRowsTestData];
        int rowIndex = 0;

        try {
            theStatement = dBConnection.prepareStatement(
                    "SELECT t1.id, t1.ch_shdr_id,"
                            + " t1.ch_id, t1.real_value, t1.binary_string"
                            + " FROM cluster_assorted t1 WHERE t1.ch_shdr_id = ?"
                            + " AND t1.ch_id = ?");
            theStatement.setInt(1, selfHdrIdIn);
            theStatement.setInt(2, clusterHdrIdIn);
            theResultSet = theStatement.executeQuery();

            // Fetch and Populate the cursor with ClusterAssorted
            while (theResultSet.next()) {
                ClusterAssorted clusterAssortedRecord = new ClusterAssorted(
                        theResultSet.getInt(1), theResultSet.getInt(2),
                        theResultSet.getInt(3), theResultSet.getInt(4),
                        theResultSet.getString(5));

                clusterAssortedArray[rowIndex] = clusterAssortedRecord;
                rowIndex++;
            }
            theResultSet.close();
            theStatement.close();
        } catch (SQLException sqle) {
            System.err.println(sqle);
        }
        return clusterAssortedArray;
    }

    /** Get clusterAssortedId
     */

    /**
     * @return clusterAssortedId
     */
    public final int getClusterAssortedId() {
        return clusterAssortedId;
    }

    /** Get selfHdrId
     */

    /**
     * @return selfHdrId
     */
    public final int getSelfHdrId() {
        return selfHdrId;
```

```
    }

    /** Get clusterHdrId
     */

    /**
     * @return clusterHdrId
     */
    public final int getClusterHdrId() {
        return clusterHdrId;
    }

    /** Get realValue
     */

    /**
     * @return realValue
     */
    public final int getRealValue() {
        return realValue;
    }

    /** Get binaryString
     */

    /**
     * @return binaryString
     */
    public final String getBinaryString() {
        return binaryString;
    }

    /** Insert cluster_assorted row.
     */
    private void insertClusterAssortedRow() {
        try {

            // Primary Key (id, ch_shdr_id, ch_id)
            theStatement = dBConnection.prepareStatement(
                    "INSERT INTO cluster_assorted"
                            + " (id, ch_shdr_id, ch_id, binary_string,"
                            + " real_value, creation_date, last_update_date)"
                            + " VALUES (?,?,?,?,?, sysdate, sysdate)");
            theStatement.setInt(1, this.clusterAssortedId);
            theStatement.setInt(2, this.selfHdrId);
            theStatement.setInt(3, this.clusterHdrId);
            theStatement.setString(4, this.binaryString);
            theStatement.setInt(3, this.realValue);
            theStatement.executeUpdate();
            theStatement.close();
        } catch (SQLException sqle) {
            System.err.println(sqle);
            System.out.println("Error during Insert");
        }
    }
}
```

Appendix O – Java Source Code for Scenario.java

```
/*
 * @(#)Scenario.java 1.00 09/05/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


/**
 * Scenario class
 */

/**
 * @author Luis Gonzalez
 * @version 1.4.2
 */
public class Scenario {

    // Class variables
    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;
    private static ResultSet theResultSet = null;

    // Instance variables
    private int scenarioId;
    private int selfHdrId;
    private int clusterHdrId;
    private double matchingProbability;
    private double probOfNotMatchingSelf;
    private int initialPopulation;
    private int repertoireSize;
    private int alphabetSize;
    private int matchingThreshold;
    private int stringLength;
    private int variationType;
    private int numberOfTrials;
    private int totalRowsSelfData;
    private int totalRowsTestData;
    private int totalNonSelfRowsInTest;
    private int maxRealValue;
    private int matchingRule;
    private String approach;
    private double meanTrueDetection;
    private double stdevTrueDetection;
    private double meanFalseDetection;
    private double stdevFalseDetection;
    private double meanTruePositive = 0;
    private double stdevTruePositive = 0;
    private double meanTrueNegative = 0;
    private double stdevTrueNegative = 0;
    private double meanFalsePositive = 0;
    private double stdevFalsePositive = 0;
    private double meanFalseNegative = 0;
    private double stdevFalseNegative = 0;

    /**
     * Initialize class with zero values for the numbers and null values for the strings.
     */
    public Scenario() {
        scenarioId = 0;
        selfHdrId = 0;
```

```
        clusterHdrId = 0;
        repertoireSize = 0;
        probOfNotMatchingSelf = 0;
        matchingThreshold = 0;
        matchingRule = 0;
        variationType = 0;
        numberOfTrials = 0;
        alphabetSize = 0;
        stringLength = 0;
        matchingProbability = 0;
        initialPopulation = 0;
        totalRowsSelfData = 0;
        totalRowsTestData = 0;
        totalNonSelfRowsInTest = 0;
        maxRealValue = 0;
        approach = null;
        meanTrueDetection = 0;
        stdevTrueDetection = 0;
        meanFalseDetection = 0;
        stdevFalseDetection = 0;
        meanTruePositive = 0;
        stdevTruePositive = 0;
        meanTrueNegative = 0;
        stdevTrueNegative = 0;
        meanFalsePositive = 0;
        stdevFalsePositive = 0;
        meanFalseNegative = 0;
        stdevFalseNegative = 0;
        return;
    }

    /**
     * Initialize class using parameters.
     *
     * @param selfHdrIdIn             The self header unique identifier.
     * @param clusterHdrIdIn          The cluster header unique identifier.
     * @param repertoireSizeIn        The repertoire size (Use 0 for automatic
calculation).
     * @param probOfNotMatchingSelfIn The probability of not matching self.
     * @param matchingThresholdIn     The matching threshold value.
     * @param matchingRuleIn          The matching rule.
     * @param variationTypeIn         The type of variation.
     * @param numberOfTrialsIn        The number of trials.
     * @param alphabetSizeIn          The alphabet size.
     * @param approachIn              The type of approach.
     */
    public Scenario(final int selfHdrIdIn, final int clusterHdrIdIn, int
repertoireSizeIn, double probOfNotMatchingSelfIn, final int matchingThresholdIn, final
int matchingRuleIn, final int variationTypeIn, final int numberOfTrialsIn, final int
alphabetSizeIn, final String approachIn) {
        scenarioId = 0;
        selfHdrId = selfHdrIdIn;
        clusterHdrId = clusterHdrIdIn;
        repertoireSize = repertoireSizeIn;
        probOfNotMatchingSelf = probOfNotMatchingSelfIn;
        matchingThreshold = matchingThresholdIn;
        matchingRule = matchingRuleIn;
        variationType = variationTypeIn;
        numberOfTrials = numberOfTrialsIn;
        alphabetSize = alphabetSizeIn;
        stringLength = 0;
        matchingProbability = 0;
        initialPopulation = 0;
        totalRowsSelfData = 0;
        totalRowsTestData = 0;
        totalNonSelfRowsInTest = 0;
        maxRealValue = 0;
        approach = approachIn;
        meanTrueDetection = 0;
        stdevTrueDetection = 0;
        meanFalseDetection = 0;
```

```
    stdevFalseDetection = 0;
    meanTruePositive = 0;
    stdevTruePositive = 0;
    meanTrueNegative = 0;
    stdevTrueNegative = 0;
    meanFalsePositive = 0;
    stdevFalsePositive = 0;
    meanFalseNegative = 0;
    stdevFalseNegative = 0;
    return;
}

/**
 * Set dBConnection.
 *
 * @param dBConnectionIn The database connection.
 */
public static void setDBConnection(final Connection dBConnectionIn) {
    dBConnection = dBConnectionIn;
}

/**
 * Create Scenario.
 */
public void createScenario() {

    // Set Next Scenario
    this.setNextScenarioId();

    // TotalRowsSelf & MaxRealValue
    this.setTotalRowsSelfData();
    System.out.println("Total Self: " + this.totalRowsSelfData);

    // Determine binary string length
    this.setStringLength();

    // Calculate Matching Probability
    this.setMatchingProbability();
    System.out.println("Matching Probability: " + this.matchingProbability);

    // Calculate Probability of Not Matching Self
    this.setProbOfNotMatchingSelf();
    System.out.println(
            "Probability of Not Matching Self: "
                    + this.probOfNotMatchingSelf);

    // Calculate Initial Population
    this.setInitialPopulation();
    System.out.println("Initial Population: " + this.initialPopulation);

    // Set (calculate) Repertoire Size
    this.setRepertoireSize();
    System.out.println("Repertoire Size: " + this.repertoireSize);

    // TotalRowsTestData & TotalNonSelfRowsInTest
    this.setTotalNonSelfRowsInTest();
    this.setTotalRowsTestData();

    // Insert Scenario
    this.insertScenarioRow();
}

/**
 * Set Statistics.
 */
public void setStatistics() {
    System.out.println();

    // Calculation of mean true detection
    this.setMeanTrueDetection();
```

```
    // Calculation of standard deviation true detection
    this.setStdevTrueDetection();

    // Calculation of mean false detection
    this.setMeanFalseDetection();

    // Calculation of standard deviation false detection
    this.setStdevFalseDetection();
    System.out.println("Statistics by Scenario \n");
    System.out.println(
            "TD: AVG = " + this.getMeanTrueDetection() + "; STDEV = "
            + this.getStdevTrueDetection() + ";\n " + "T+: AVG = "
            + this.getMeanTruePositive() + "; STDEV = "
            + this.getStdevTruePositive() + ";\n " + "T-: AVG = "
            + this.getMeanTrueNegative() + "; STDEV = "
            + this.getStdevTrueNegative() + "\n");
    System.out.println(
            "FD: AVG = " + this.getMeanFalseDetection() + "; STDEV = "
            + this.getStdevFalseDetection() + ";\n " + "F+: AVG = "
            + this.getMeanFalsePositive() + "; STDEV = "
            + this.getStdevFalsePositive() + ";\n " + "F-: AVG = "
            + this.getMeanFalseNegative() + "; STDEV = "
            + this.getStdevFalseNegative() + "\n");
    this.updateScenarioRow();
}

/**
 * Set nextScenarioId.
 */
private void setNextScenarioId() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT (nvl(max(id),0)+1) FROM scenarios"
                        + " WHERE ch_shdr_id = ? and ch_id = ?");
        theStatement.setInt(1, selfHdrId);
        theStatement.setInt(2, clusterHdrId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            scenarioId = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the scenario unique identifier
 */

/**
 * @return The scenario unique identifier
 */
public int getScenarioId() {
    return scenarioId;
}

/**
 * Set string Length
 */
private void setStringLength() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT t1.binary_string_length"
                        + " FROM self_header t1 WHERE t1.id = ?");
        theStatement.setInt(1, selfHdrId);
        theResultSet = theStatement.executeQuery();
```

```
        // Fetch the cursor
        while (theResultSet.next()) {
            stringLength = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the string length.
 */

/**
 * @return The string length
 */
public int getStringLength() {
    return stringLength;
}

/**
 * Set matchingProbability (Pm).
 */
private void setMatchingProbability() {

    // Pm = m ^ -r [ (l - r) * (m - 1) / m + 1 ]
    try {
        this.matchingProbability = 1
                / Math.pow(this.alphabetSize, this.matchingThreshold)
                * ((double) (this.stringLength - this.matchingThreshold)
                        * (this.alphabetSize - 1) / alphabetSize
                                + 1);
    } catch (Exception e) {
        System.err.println(e);
    }

    // this.matchingProbability = (double) 1 / this.totalRowsSelfData;
}

/**
 * Set Probability of Failure (Pf).
 */
private void setProbOfNotMatchingSelf() {
    try {

        // Pf = Probability that a detector fails to match a non-self

        // Pf = (1-Pm) ^ NR
        if (this.probOfNotMatchingSelf == 0) {
            this.probOfNotMatchingSelf = (double) Math.pow(
                    (1 - this.matchingProbability), this.totalRowsSelfData);
        }
    } catch (Exception e) {
        System.err.println(e);
    }
}

/**
 * Set initial population of detectors before negative selection.
 */
private void setInitialPopulation() {
    try {

        // NR0 = -ln(Pf) / Pm (1 - Pm) ^ Ns
        double initialPopulationDouble = (double) -Math.log(
                this.probOfNotMatchingSelf)
                        / (this.matchingProbability
                                * Math.pow((1 - this.matchingProbability),
                                this.totalRowsSelfData));
```

```
        this.initialPopulation = (int) initialPopulationDouble;
    } catch (Exception e) {
        System.err.println(e);
    }
}

/**
 * Returns the initial population
 */

/**
 * @return The initial population
 */
public int getInitialPopulation() {
    return initialPopulation;
}

/**
 * Set repertoireSize Population of competent detectors after negative selection.
 */
private void setRepertoireSize() {
    try {

        // NR = -ln(Pf) / Pm
        if (this.repertoireSize == 0) {
            double repertoireSizeDouble = (double) -Math.log(
                    this.probOfNotMatchingSelf)
                            / this.matchingProbability;

            this.repertoireSize = (int) repertoireSizeDouble;
        }
    } catch (Exception e) {
        System.err.println(e);
    }
}

/**
 * Returns the repertoire size
 */

/**
 * @return The repertoire size
 */
public int getRepertoireSize() {
    return repertoireSize;
}

/**
 * Set totalRowsSelfData.
 */
private void setTotalRowsSelfData() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT COUNT(t1.real_value) FROM self t1 WHERE t1.shdr_id = ?");
        theStatement.setInt(1, selfHdrId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            totalRowsSelfData = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the total rows in the SELF table.
```

```
 *
 * @return The total rows in the SELF table.
 */
public int getTotalRowsSelfData() {
    return totalRowsSelfData;
}

/**
 * Set totalRowsTestData.
 */
private void setTotalRowsTestData() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT COUNT(t1.real_value)"
                        + " FROM cluster_assorted t1 WHERE t1.ch_id = ?"
                        + " AND t1.ch_shdr_id = ?");
        theStatement.setInt(1, clusterHdrId);
        theStatement.setInt(2, selfHdrId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            totalRowsTestData = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the total rows in the CLUSTER_ASSORTED table.
 *
 * @return The total rows in the CLUSTER_ASSORTED table.
 */
public int getTotalRowsTestData() {
    return totalRowsTestData;
}

/**
 * Set totalNonSelfRowsInTest.
 */
private void setTotalNonSelfRowsInTest() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT COUNT(t1.real_value) FROM cluster_assorted t1"
                        + " WHERE t1.ch_shdr_id = ? AND t1.ch_id = ?"
                        + " AND t1.real_value NOT IN"
                        + " (SELECT t2.real_value FROM SELF t2"
                        + " WHERE t2.shdr_id = t1.ch_shdr_id)");
        theStatement.setInt(1, selfHdrId);
        theStatement.setInt(2, clusterHdrId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            totalNonSelfRowsInTest = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
        System.out.println(
                "Nonself In Cluster Assorted: " + totalNonSelfRowsInTest);
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the total of nonself rows in the CLUSTER_ASSORTED table.
 *
```

```
 * @return The total of nonself rows in the CLUSTER_ASSORTED table.
 */
public int getTotalNonSelfRowsInTest() {
    return totalNonSelfRowsInTest;
}

/**
 * Set maxRealValue.
 */
private void setMaxRealValue() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT max(to_number(t1.real_value)) FROM self t1"
                        + " WHERE t1.shdr_id = ?");
        theStatement.setInt(1, selfHdrId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            this.maxRealValue = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Set mean true detection.
 */
private void setMeanTrueDetection() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT AVG ( t1.total_true_positive"
                        + " + t1.total_true_negative),"
                        + " AVG ( t1.total_true_positive ),"
                        + " AVG ( t1.total_true_negative )"
                        + " FROM trials t1"
                        + " WHERE t1.scnr_ch_shdr_id = ? AND"
                        + " t1.scnr_ch_id = ? AND t1.scnr_id = ?");
        theStatement.setInt(1, this.selfHdrId);
        theStatement.setInt(2, this.clusterHdrId);
        theStatement.setInt(3, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            this.meanTrueDetection = theResultSet.getDouble(1);
            this.meanTruePositive = theResultSet.getDouble(2);
            this.meanTrueNegative = theResultSet.getDouble(3);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the mean of true detections.
 *
 * @return The mean of true detections.
 */
public double getMeanTrueDetection() {
    return meanTrueDetection;
}

/**
 * Set standard deviation true detection (true positive + true negative).
 */
```

```
private void setStdevTrueDetection() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT SQRT (SUM (POWER ((t1.total_true_positive"
                        + " + t1.total_true_negative)- ?, 2))"
                        + " / (  ? - 1)),"
                        + " SQRT (SUM (POWER (t1.total_true_positive"
                        + " - ? , 2)) / (  ? - 1)),"
                        + " SQRT (SUM (POWER (t1.total_true_negative"
                        + " - ?,2)) / (  ? - 1)) FROM trials t1"
                        + " WHERE t1.scnr_ch_shdr_id = ? AND"
                        + " t1.scnr_ch_id = ? AND t1.scnr_id = ?");
        theStatement.setDouble(1, this.meanTrueDetection);
        theStatement.setInt(2, this.numberOfTrials);
        theStatement.setDouble(3, this.meanTrueDetection);
        theStatement.setInt(4, this.numberOfTrials);
        theStatement.setDouble(5, this.meanTrueDetection);
        theStatement.setInt(6, this.numberOfTrials);
        theStatement.setInt(7, this.selfHdrId);
        theStatement.setInt(8, this.clusterHdrId);
        theStatement.setInt(9, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            this.stdevTrueDetection = theResultSet.getDouble(1);
            this.stdevTruePositive = theResultSet.getDouble(2);
            this.stdevTrueNegative = theResultSet.getDouble(3);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the standard deviation of true detection.
 *
 * @return The standard deviation of true detections.
 */
public double getStdevTrueDetection() {
    return stdevTrueDetection;
}

/**
 * Set mean false detection.
 */
private void setMeanFalseDetection() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT AVG (t1.total_false_positive"
                        + " + t1.total_false_negative),"
                        + "AVG ( t1.total_false_positive ),"
                        + " AVG ( t1.total_false_negative )"
                        + " FROM trials t1 WHERE"
                        + " t1.scnr_ch_shdr_id = ? AND t1.scnr_ch_id = ?"
                        + " AND t1.scnr_id = ?");
        theStatement.setInt(1, this.selfHdrId);
        theStatement.setInt(2, this.clusterHdrId);
        theStatement.setInt(3, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            this.meanFalseDetection = theResultSet.getDouble(1);
            this.meanFalsePositive = theResultSet.getDouble(2);
            this.meanFalseNegative = theResultSet.getDouble(3);
        }
        theResultSet.close();
        theStatement.close();
```

```
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the mean of false detections.
 *
 * @return The mean of false detections.
 */
public double getMeanFalseDetection() {
    return meanFalseDetection;
}

/**
 * Set standard deviation false detection (false positive + false negative).
 */
private void setStdevFalseDetection() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT SQRT (SUM (POWER ((t1.total_false_positive"
                        + " + t1.total_false_negative)- ?,2)) / (  ? - 1)),"
                        + " SQRT (SUM (POWER (t1.total_false_positive"
                        + " - ?,2)) / (  ? - 1)),"
                        + " SQRT (SUM (POWER (t1.total_false_negative"
                        + " - ?,2)) / (  ? - 1))"
                        + " FROM trials t1 WHERE t1.scnr_ch_shdr_id = ?"
                        + " AND t1.scnr_ch_id = ? AND t1.scnr_id = ?");
        theStatement.setDouble(1, this.meanFalseDetection);
        theStatement.setInt(2, this.numberOfTrials);
        theStatement.setDouble(3, this.meanFalseDetection);
        theStatement.setInt(4, this.numberOfTrials);
        theStatement.setDouble(5, this.meanFalseDetection);
        theStatement.setInt(6, this.numberOfTrials);
        theStatement.setInt(7, this.selfHdrId);
        theStatement.setInt(8, this.clusterHdrId);
        theStatement.setInt(9, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor to obtain scenarioId
        while (theResultSet.next()) {
            this.stdevFalseDetection = theResultSet.getDouble(1);
            this.stdevFalsePositive = theResultSet.getDouble(2);
            this.stdevFalseNegative = theResultSet.getDouble(3);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
    }
}

/**
 * Returns the standard deviation of false detections.
 *
 * @return The standard deviation of false detections.
 */
public double getStdevFalseDetection() {
    return stdevFalseDetection;
}

/**
 * Returns the mean of true positive.
 *
 * @return The mean of true positive.
 */
public double getMeanTruePositive() {
    return meanTruePositive;
}

/**
```

```
 * Returns the standard deviation of true positive.
 *
 * @return The standard deviation of true positive.
 */
public double getStdevTruePositive() {
    return stdevTruePositive;
}

/**
 * Returns the mean of true negative.
 *
 * @return The mean of true negative.
 */
public double getMeanTrueNegative() {
    return meanTrueNegative;
}

/**
 * Returns the standard deviation of true negative.
 *
 * @return The standard deviation of true negative.
 */
public double getStdevTrueNegative() {
    return stdevTrueNegative;
}

/**
 * Returns the mean of false positive.
 *
 * @return The mean of false positive.
 */
public double getMeanFalsePositive() {
    return meanFalsePositive;
}

/**
 * Returns the standard deviation of false positive.
 *
 * @return The standard deviation of false positive.
 */
public double getStdevFalsePositive() {
    return stdevFalsePositive;
}

/**
 * Returns the mean of false negative.
 *
 * @return The mean of false negative.
 */
public double getMeanFalseNegative() {
    return meanFalseNegative;
}

/**
 * Returns the standard deviation of false negative.
 *
 * @return The standard deviation of false negative.
 */
public double getStdevFalseNegative() {
    return stdevFalseNegative;
}

/**
 * Insert Scenario.
 */
private void insertScenarioRow() {
    try {
        theStatement = dBConnection.prepareStatement(
                "INSERT INTO scenarios"
                        + " (id, ch_shdr_id, ch_id, matching_probability,"
                        + " repertoire_size, probability_failure,"
```

```
                            + " initial_population, matching_threshold,"
                            + " variation_type, number_of_trials,"
                            + " matching_rule, approach,"
                            + " total_self, total_cluster,"
                            + " total_nonself_in_cluster,"
                            + " creation_date, last_update_date) VALUES"
                            + " (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,"
                            + " sysdate, sysdate)");
            theStatement.setInt(1, this.scenarioId);
            theStatement.setInt(2, this.selfHdrId);
            theStatement.setInt(3, this.clusterHdrId);
            theStatement.setDouble(4, this.matchingProbability);
            theStatement.setInt(5, this.repertoireSize);
            theStatement.setDouble(6, this.probOfNotMatchingSelf);
            theStatement.setInt(7, this.initialPopulation);
            theStatement.setInt(8, this.matchingThreshold);
            theStatement.setInt(9, this.variationType);
            theStatement.setInt(10, this.numberOfTrials);
            theStatement.setInt(11, this.matchingRule);
            theStatement.setString(12, this.approach);
            theStatement.setInt(13, this.totalRowsSelfData);
            theStatement.setInt(14, this.totalRowsTestData);
            theStatement.setInt(15, this.totalNonSelfRowsInTest);
            theStatement.executeUpdate();
            theStatement.close();
        } catch (SQLException sqle) {
            System.err.println(sqle);
        }
    }

    /**
     * Update scenario row.
     */
    private void updateScenarioRow() {
        try {

            // PRIMARY KEY (id, ch_schr_id, ch_id)
            theStatement = dBConnection.prepareStatement(
                    "UPDATE scenarios t1 SET t1.mean_true_detection = ? ,"
                            + " t1.stdev_true_detection = ? ,"
                            + " t1.mean_false_detection = ? ,"
                            + " t1.stdev_false_detection = ? ,"
                            + " t1.last_update_date = sysdate"
                            + " WHERE t1.id = ? and t1.ch_shdr_id = ?"
                            + " and t1.ch_id = ?");
            theStatement.setDouble(1, this.meanTrueDetection);
            theStatement.setDouble(2, this.stdevTrueDetection);
            theStatement.setDouble(3, this.meanFalseDetection);
            theStatement.setDouble(4, this.stdevFalseDetection);
            theStatement.setInt(5, this.scenarioId);
            theStatement.setInt(6, this.selfHdrId);
            theStatement.setInt(7, this.clusterHdrId);
            theStatement.executeUpdate();
            theStatement.close();
        } catch (SQLException sqle) {
            System.err.println(sqle);
            System.out.println("Error during Update");

            // loop through exceptions
            do {
                System.err.println(
                        "Exception ocurred:\nMessage: " + sqle.getMessage());
                System.err.println("SQL state: " + sqle.getSQLState());
                System.err.println(
                        "Vendor code: " + sqle.getErrorCode()
                        + "\n---------------");
            }while ((sqle = sqle.getNextException()) != null);
        }
    }
}
```

Appendix P – Java Source Code for Trial.java

```
/*
 * @(#)Trial.java 1.00 09/05/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


/** Trial class
 *
 * @author  Luis Gonzalez
 * @version 1.4.2
 */
public class Trial {

    // Class variables
    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;
    private static ResultSet theResultSet = null;

    // Instance variables
    private int trialId;
    private int scenarioId;
    private int selfHdrId;
    private int clusterHdrId;
    private int totalDetection;
    private int totalTruePositive;
    private int totalTrueNegative;
    private int totalFalsePositive;
    private int totalFalseNegative;
    private long trialDuration;
    private int matchingRule;

    /**
     * Initialize class with zero values.
     */
    public Trial() {
        trialId = 0;
        scenarioId = 0;
        selfHdrId = 0;
        clusterHdrId = 0;
        totalDetection = 0;
        totalTruePositive = 0;
        totalTrueNegative = 0;
        totalFalsePositive = 0;
        totalFalseNegative = 0;
        trialDuration = 0;
        matchingRule = 0;
        return;
    }

    /**
     * Initialize class with parameters.
     */
    public Trial(final int trialIdIn, final int scenarioIdIn, final int selfHdrIdIn,
final int clusterHdrIdIn, final int matchingRuleIn) {
        trialId = trialIdIn;
        scenarioId = scenarioIdIn;
        selfHdrId = selfHdrIdIn;
        clusterHdrId = clusterHdrIdIn;
        totalDetection = 0;
```

```
    totalTruePositive = 0;
    totalTrueNegative = 0;
    totalFalsePositive = 0;
    totalFalseNegative = 0;
    trialDuration = 0;
    matchingRule = matchingRuleIn;
    return;
}

/**
 * Set dBConnection.
 *
 * @param dBConnectionIn Database Connection
 */
public static void setDBConnection(final Connection dBConnectionIn) {
    dBConnection = dBConnectionIn;
}

/**
 * Set trialId
 *
 * @param trialIdIn Trial Unique Identifier (Cycle Id)
 */
public final void setTrialId(final int trialIdIn) {
    trialId = trialIdIn;
}

/**
 * Get trialId
 *
 * @return Trial Unique Identifier
 */
public final int getTrialId() {
    return trialId;
}

/**
 * Set scenarioId
 *
 * @param scenarioIdIn Scenario Unique Identifier
 */
public final void setScenarioId(final int scenarioIdIn) {
    scenarioId = scenarioIdIn;
}

/**
 * Get scenarioId
 *
 * @return scenarioId Scenario Unique Identifier
 */
public final int getScenarioId() {
    return scenarioId;
}

/**
 * Set selfHdrId
 *
 * @param selfHdrIdIn Self Header Unique Identifier
 */
public final void setSelfHdrId(final int selfHdrIdIn) {
    selfHdrId = selfHdrIdIn;
}

/**
 * Get selfHdrId
 *
 * @return selfHdrId Self Header Unique Identifier
 */
public final int getSelfHdrId() {
    return selfHdrId;
}
```

```
    /**
     * Method to set clusterHdrId
     *
     * @param clusterHdrIdIn Cluster Header Unique Identifier
     */
    public final void setClusterHdrId(final int clusterHdrIdIn) {
        clusterHdrId = clusterHdrIdIn;
    }

    /**
     * Get clusterHdrId
     *
     * @return clusterHdrId Cluster Header Unique Identifier
     */
    public final int getClusterHdrId() {
        return clusterHdrId;
    }

    /**
     * Set totalDetection.
     *
     * @param  totalDetectionIn Total Detection
     */
    public final void setTotalDetection(final int totalDetectionIn) {
        totalDetection = totalDetectionIn;
    }

    /**
     * Monitor (examine) test data with competent detectors.
     *
     * @param totalRowsTestDataIn The total rows in the CLUSTER_ASSORTED table.
     * @param repertoireSizeIn    The repertoire size unique identifier.
     * @param matchingThresholdIn The matching treshold value.
     * @param testDataset         The dataset to be examined.
     * @param competentDetectors  The set of competent detectors that will inspect the
dataset.
     */
    public final void setTotalDetection(final int totalRowsTestDataIn, final int
repertoireSizeIn, final int matchingThresholdIn, final ClusterAssorted[] testDataset,
final Detector[] competentDetectors) {
        int totalRowsTestData = totalRowsTestDataIn;
        int repertoireSize = repertoireSizeIn;
        int matchingThreshold = matchingThresholdIn;
        int affinity = 0;
        int x = 1;
        String xorOutput = null;
        NonSelf nonSelfRecord = new NonSelf();

        NonSelf.setdBConnection(dBConnection);
        for (int rowIndexTestData = 0; rowIndexTestData < totalRowsTestData;
rowIndexTestData++) {
            for (int rowIndex = 0; rowIndex < repertoireSize; rowIndex++) {

                // Calculate affinity between Test Data and Detector
                xorOutput = Long.toBinaryString(
                        testDataset[rowIndexTestData].getRealValue()
                                ^ competentDetectors[rowIndex].getRealValue());
                MatchingRule matchingRuleRecord = new MatchingRule(
                        this.matchingRule, xorOutput);

                affinity = matchingRuleRecord.getDistance();

                //
                if (affinity >= matchingThreshold) {

                    // Insert NonSelf
                    nonSelfRecord = new NonSelf(x++, this.trialId,
                            this.scenarioId, this.clusterHdrId, this.selfHdrId,
                            testDataset[rowIndexTestData].getRealValue(),
                            testDataset[rowIndexTestData].getBinaryString());
```

```
                nonSelfRecord.createNonSelf();

                // System.out.println(testDataset[rowIndexTestData].getRealValue()

                // + " ("

                // + testDataset[rowIndexTestData].getBinaryString()

                // + ") has been identified as NONSELF data");
                totalDetection = totalDetection + 1;
                break;
            }
        }
    }
}

/**
 * Get totalDetection
 *
 * @return totalDetection Total Detection
 */
public final int getTotalDetection() {
    return totalDetection;
}

/**
 * Set totalTruePositive.
 */
public final void setTotalTruePositive() {

    /* The results of this query will be used to obtain
     * the rate at which non-self is correctly detected
     */
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT NVL (COUNT (t1.real_value), 0)"
                        + " FROM cluster_assorted t1"
                        + " WHERE t1.ch_shdr_id = ? AND t1.ch_id = ?"
                        + " AND t1.real_value NOT IN (SELECT t2.real_value"
                        + " FROM self t2"
                        + " WHERE t2.real_value = t1.real_value)"
                        + " AND t1.real_value IN (SELECT t3.real_value"
                        + " FROM nonself t3 WHERE t3.trial_id = ?"
                        + " AND t3.trial_scnr_ch_shdr_id = t1.ch_shdr_id"
                        + " AND t3.trial_scnr_ch_id = t1.ch_id"
                        + " AND t3.trial_scnr_id = ?"
                        + " AND t3.real_value = t1.real_value)");
        theStatement.setInt(1, this.selfHdrId);
        theStatement.setInt(2, this.clusterHdrId);
        theStatement.setInt(3, this.trialId);
        theStatement.setInt(4, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            this.totalTruePositive = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
        System.out.println("Error when setting True Positive");
    }
}

/**
 * Get totalTruePositive
 *
 * @return totalTruePositive Total True Positive
 */
public final int getTotalTruePositive() {
```

```
    return totalTruePositive;
}

/**
 * Set totalTrueNegative
 */

/*
 * The results of this query will be used to obtain the
 * rate at which self is correctly not detected
 */
public final void setTotalTrueNegative() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT NVL (COUNT (t1.real_value), 0)"
                        + " FROM cluster_assorted t1"
                        + " WHERE t1.ch_shdr_id = ? AND t1.ch_id = ?"
                        + " AND t1.real_value IN (SELECT t2.real_value"
                        + " FROM self t2"
                        + " WHERE t2.real_value = t1.real_value)"
                        + " AND t1.real_value NOT IN (SELECT t3.real_value"
                        + " FROM nonself t3 WHERE t3.trial_id = ?"
                        + " AND t3.trial_scnr_ch_shdr_id = t1.ch_shdr_id"
                        + " AND t3.trial_scnr_ch_id = t1.ch_id"
                        + " AND t3.trial_scnr_id = ?"
                        + " AND t3.real_value = t1.real_value)");
        theStatement.setInt(1, this.selfHdrId);
        theStatement.setInt(2, this.clusterHdrId);
        theStatement.setInt(3, this.trialId);
        theStatement.setInt(4, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            this.totalTrueNegative = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
        System.out.println("Error when setting True Negative");
    }
}

/**
 * Get totalTrueNegative
 *
 * @return totalTrueNegative Total True Negative
 */
public final int getTotalTrueNegative() {
    return totalTrueNegative;
}

/**
 * Set totalFalsePositive
 */

/* The results of this query will be used to obtain the
 * rate at which self is incorrectly detected
 */
public final void setTotalFalsePositive() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT NVL (COUNT (t1.real_value), 0)"
                        + " FROM cluster_assorted t1"
                        + " WHERE t1.ch_shdr_id = ? AND t1.ch_id = ?"
                        + " AND t1.real_value IN (SELECT t2.real_value"
                        + " FROM self t2"
                        + " WHERE t2.real_value = t1.real_value)"
                        + " AND t1.real_value IN (SELECT t3.real_value"
                        + " FROM nonself t3 WHERE t3.trial_id = ?"
```

```
                        + " AND t3.trial_scnr_ch_shdr_id = t1.ch_shdr_id"
                        + " AND t3.trial_scnr_ch_id = t1.ch_id"
                        + " AND t3.trial_scnr_id = ?"
                        + " AND t3.real_value = t1.real_value)");
        theStatement.setInt(1, this.selfHdrId);
        theStatement.setInt(2, this.clusterHdrId);
        theStatement.setInt(3, this.trialId);
        theStatement.setInt(4, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            this.totalFalsePositive = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
        System.out.println("Error when setting False Positive");
    }
}

/**
 * Get totalFalsePositive
 *
 * @return totalFalsePositive Total False Positive
 */
public final int getTotalFalsePositive() {
    return totalFalsePositive;
}

/**
 * Set totalFalseNegative
 */

/* The results of this query will be used to obtain the
 * rate at which non-self is not detected
 */
public final void setTotalFalseNegative() {
    try {
        theStatement = dBConnection.prepareStatement(
                "SELECT NVL (COUNT (t1.real_value), 0)"
                        + " FROM cluster_assorted t1"
                        + " WHERE t1.ch_shdr_id = ? AND t1.ch_id = ?"
                        + " AND t1.real_value NOT IN (SELECT t2.real_value"
                        + " FROM self t2"
                        + " WHERE t2.real_value = t1.real_value)"
                        + " AND t1.real_value NOT IN (SELECT t3.real_value"
                        + " FROM nonself t3 WHERE t3.trial_id = ?"
                        + " AND t3.trial_scnr_ch_shdr_id = t1.ch_shdr_id"
                        + " AND t3.trial_scnr_ch_id = t1.ch_id"
                        + " AND t3.trial_scnr_id = ?"
                        + " AND t3.real_value = t1.real_value)");
        theStatement.setInt(1, this.selfHdrId);
        theStatement.setInt(2, this.clusterHdrId);
        theStatement.setInt(3, this.trialId);
        theStatement.setInt(4, this.scenarioId);
        theResultSet = theStatement.executeQuery();

        // Fetch the cursor
        while (theResultSet.next()) {
            this.totalFalseNegative = theResultSet.getInt(1);
        }
        theResultSet.close();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
        System.out.println("Error when setting False Negative");
    }
}
```

```
/**
 * Get totalFalseNegative
 *
 * @return totalFalsePositive Total False Positive
 */
public final int getTotalFalseNegative() {
    return totalFalsePositive;
}

/**
 * Set trialDuration
 *
 * @param  trialDurationIn (milliseconds)
 */
public final void setTrialDuration(final long trialDurationIn) {
    trialDuration = trialDurationIn;
}

/** Get trialDuration
 *
 * @return trialDuration Trial Duration
 */
public final long getTrialDuration() {
    return trialDuration;
}

/**
 * Insert trial row.
 */
public final void insertTrialRow() {
    try {
        theStatement = dBConnection.prepareStatement(
                "INSERT INTO trials"
                        + " (id, scnr_ch_shdr_id, scnr_ch_id, scnr_id,"
                        + " creation_date, last_update_date)"
                        + " VALUES (?,?,?,?, sysdate, sysdate)");
        theStatement.setInt(1, this.trialId);
        theStatement.setInt(2, this.selfHdrId);
        theStatement.setInt(3, this.clusterHdrId);
        theStatement.setInt(4, this.scenarioId);
        theStatement.executeUpdate();
        theStatement.close();
    } catch (SQLException sqle) {
        System.err.println(sqle);
        System.out.println("Error during Insert");
    }
}

/**
 * Update trial row.
 */
public final void updateTrialRow() {
    try {

        // PRIMARY KEY (id, scnr_id, scnr_ch_shdr_id, scnr_ch_id)
        theStatement = dBConnection.prepareStatement(
                "UPDATE trials t1 SET t1.duration = ? ,"
                        + " t1.total_detection = ? ,"
                        + " t1.total_true_positive = ? ,"
                        + " t1.total_true_negative = ? ,"
                        + " t1.total_false_positive = ? ,"
                        + " t1.total_false_negative = ?, "
                        + " t1.last_update_date = sysdate"
                        + " WHERE t1.id = ? and t1.scnr_id = ?"
                        + " and t1.scnr_ch_shdr_id = ?"
                        + " and t1.scnr_ch_id = ?");
        theStatement.setDouble(1, this.trialDuration);
        theStatement.setInt(2, this.totalDetection);
        theStatement.setInt(3, this.totalTruePositive);
        theStatement.setInt(4, this.totalTrueNegative);
        theStatement.setInt(5, this.totalFalsePositive);
```

```
            theStatement.setInt(6, this.totalFalseNegative);
            theStatement.setInt(7, this.trialId);
            theStatement.setInt(8, this.scenarioId);
            theStatement.setInt(9, this.selfHdrId);
            theStatement.setInt(10, this.clusterHdrId);
            theStatement.executeUpdate();
            theStatement.close();
        } catch (SQLException sqle) {
            System.err.println(sqle);
            System.out.println("Error during Update");

            // loop through exceptions
            do {
                System.err.println(
                        "Exception ocurred:\nMessage: " + sqle.getMessage());
                System.err.println("SQL state: " + sqle.getSQLState());
                System.err.println(
                        "Vendor code: " + sqle.getErrorCode()
                        + "\n---------------");
            }while ((sqle = sqle.getNextException()) != null);
        }
    }

    /**
     * Display Statistics
     */
    public void displayStatistics() {
        System.out.println(
                "T/Match = " + this.getTotalDetection() + "; TD = "
                + (this.getTotalTruePositive() + this.getTotalTrueNegative())
                + "; FD = "
                + (this.getTotalFalsePositive() + this.getTotalFalseNegative()));
    }
}
```

Appendix Q – Java Source Code for Detector.java

```
/*
 * @(#)Detector.java 1.00 09/05/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Arrays;
import java.util.Random;


/**
 * Detector (Antibody) class
 *
 * @author  Luis Gonzalez
 * @version  1.4.2
 */
public class Detector {

    // Class variables
    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;
    private static ResultSet theResultSet = null;

    // Instance variables
    private int detectorId;
    private int selfHdrId;
    private int clusterHdrId;
    private int scenarioId;
    private int trialId;
    private int realValue;
    private int affinity;
    private String binaryString;
    private double mutationStepSize;

    /**
     * Constructor to initialize class.
     */
    public Detector() {
        detectorId = 0;
        selfHdrId = 0;
        clusterHdrId = 0;
        scenarioId = 0;
        trialId = 0;
        realValue = 0;
        affinity = 0;
        binaryString = null;
        mutationStepSize = 0;
        return;
    }

    /**
     * Constructor to initialize class.
     */
    public Detector(final int detectorIdIn, final int selfHdrIdIn, final int
clusterHdrIdIn, final int scenarioIdIn, final int trialIdIn, final int realValueIn, final
int affinityMeanIn, final String binaryStringIn, final double mutationStepSizeIn) {
        detectorId = detectorIdIn;
        selfHdrId = selfHdrIdIn;
        clusterHdrId = clusterHdrIdIn;
        scenarioId = scenarioIdIn;
        trialId = trialIdIn;
```

```
        realValue = realValueIn;
        affinity = affinityMeanIn;
        binaryString = binaryStringIn;
        mutationStepSize = mutationStepSizeIn;
        return;
    }

    /**
     * Method to set dBConnection
     *
     * @param dBConnectionIn
     */
    public static void setDBConnection(final Connection dBConnectionIn) {
        dBConnection = dBConnectionIn;
    }

    /**
     * Method to Calculate Standard Deviation
     *
     * @param numberIn
     *
     * @return stdev
     */
    public static double calculateStdev(final double numberIn) {

        // Variance
        double variance = Math.pow((numberIn), 2) / 2;

        // Standard deviation
        double stdev = Math.sqrt(variance);

        return stdev;
    }

    /**
     * Method to Mutate Mutation Step Size
     *
     * @param n                     dimension of the problem,e.g.,
     *                              for an 8-bit vector the dimension would be 8
     * @param mutationStepSizeIn    mutation step size
     *
     * @return mutationStepSizeMutated
     */
    public static double mutateMutationStepSize(final int n, final double
mutationStepSizeIn) {
        Random rand = new Random();

        /* tau is usually a scalar based on the number of dimensions of the
         problem. When using one tau value, it is usually set to 1/sqrt(2n),
         where there are n dimensions in the problem.

         If you have an 8-bit vector, then 8 is the dimension. If you have a
         single integer, then 1 is the dimension.
         */
        double tau = 1 / Math.sqrt(2 * n);
        double noiseForMutationStepSize = Math.pow(Math.E,
                tau * rand.nextGaussian());
        double mutationStepSizeMutated = mutationStepSizeIn
                * noiseForMutationStepSize;

        return mutationStepSizeMutated;
    }

    /**
     * Method to Mutate Gene
     *
     * @param  geneIn                   gene
     * @param  mutationStepSizeMutated  mutation step size mutated
     *
     * @return geneMutated
     */
```

```
    public static double mutateGene(final double geneIn, final double
mutationStepSizeMutated) {
        Random rand = new Random();
        double geneMutated = geneIn
                + (mutationStepSizeMutated * rand.nextGaussian());

        return Math.abs(geneMutated);
    }

    /**
     * Find whether detector is duplicated
     *
     * @param key                the key
     * @param competentDetectors  the competent detectors
     * @param currentArraySize    the current array size
     *
     * @return flag
     */
    public static boolean findDuplicatedDetectors(final int key, final Detector[]
competentDetectors, final int currentArraySize) {
        boolean flag = false;
        int arrayLength = competentDetectors.length;
        int indexDuplicatedDetector = 0;
        int[] elements = new int[arrayLength];

        for (int rowIndex = 0; rowIndex < currentArraySize; rowIndex++) {
            elements[rowIndex] = competentDetectors[rowIndex].getRealValue();
        }
        Arrays.sort(elements);
        indexDuplicatedDetector = Arrays.binarySearch(elements, key);
        if (indexDuplicatedDetector >= 0) {
            flag = true;
        }
        return flag;
    }

    /**
     * Calculation of affinity mean of a detector
     * against entire self population (Evaluation Function)
     *
     * @param candidateDetectorInteger the candidate detector
     * @param self                     the self
     * @param matchingRule             the mathing rule
     * @param matchingThreshold        the matching threshold
     *
     * @return affinityMean
     */
    public static int calculateAffinityDetectorSelf(final int candidateDetectorInteger,
final Self[] self, final int matchingRule, final int matchingThreshold) {
        String xorOutput = null;
        int affinity = 0;
        double affinityMean = 0;
        int affinitySum = 0;

        for (int rowIndexSelf = 0; rowIndexSelf < self.length; rowIndexSelf++) {

            // Calculate affinity between random number and self
            xorOutput = Integer.toBinaryString(
                    candidateDetectorInteger ^ self[rowIndexSelf].getRealValue());
            MatchingRule matchingRuleRecord = new MatchingRule(matchingRule,
                    xorOutput);

            affinity = matchingRuleRecord.getDistance();
            affinitySum = affinitySum + affinity;
            affinityMean = (double) affinitySum / (rowIndexSelf + 1);
            if (affinity >= matchingThreshold) {
                break;
            }
        }
        return (int) affinityMean;
    }
```

```
/**
 * Method to get detectorId
 *
 * @return detectorId
 */
public final int getDetectorId() {
    return detectorId;
}

/**
 * Get selfHdrId
 *
 * @return selfHdrId
 */
public final int getSelfHdrId() {
    return selfHdrId;
}

/**
 * Get clusterHdrId
 *
 * @return clusterHdrId
 */
public final int getClusterHdrId() {
    return clusterHdrId;
}

/**
 * Method to get scenarioId
 *
 * @return scenarioId
 */
public final int getScenarioId() {
    return scenarioId;
}

/**
 * Get trialId
 *
 * @return trialId
 */
public final int getTrialId() {
    return trialId;
}

/**
 * Method to get realValue
 *
 * @return realValue
 */
public final int getRealValue() {
    return realValue;
}

/**
 * Get affinity
 *
 * @return affinity
 */
public final int getAffinity() {
    return affinity;
}

/**
 * Get binaryString
 *
 * @return binaryString
 */
public final String getBinaryString() {
    return binaryString;
```

```
    }

    /**
     * Get mutationStepSize
     *
     * @return mutationStepSize
     */
    public final double getMutationStepSize() {
        return mutationStepSize;
    }

    /**
     * Method to build Insert Detector Row.
     */
    public final void insertDetectorRow() {
        try {
            theStatement = dBConnection.prepareStatement(
                    "INSERT INTO detectors"
                            + " (id, trial_scnr_ch_shdr_id, trial_scnr_ch_id,"
                            + " trial_scnr_id, trial_id, binary_string,"
                            + " real_value, mutation_step_size, affinity,"
                            + " creation_date, last_update_date)"
                            + " VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?,"
                            + " sysdate, sysdate)");
            theStatement.setInt(1, this.detectorId);
            theStatement.setInt(2, this.selfHdrId);
            theStatement.setInt(3, this.clusterHdrId);
            theStatement.setInt(4, this.scenarioId);
            theStatement.setInt(5, this.trialId);
            theStatement.setString(6, this.binaryString);
            theStatement.setInt(7, this.realValue);
            theStatement.setDouble(8, this.mutationStepSize);
            theStatement.setInt(9, this.affinity);
            theStatement.executeUpdate();
            theStatement.close();
        } catch (SQLException sqle) {
            System.out.println("Error during INSERT");
            System.err.println(sqle);
        }
    }
}
```

Appendix R – Java Source Code for NonSelf.java

```
/*
 * @(#)NonSelf.java 1.00 09/24/03
 *
 * Copyright © 2003 by Luis J. Gonzalez.
 * E-mail: luisg@nova.edu
 * All Rights Reserved.
 *
 */

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;


/** NonSelf (Antigen) class
 */

/**
 * @author     Luis Gonzalez
 * @version    1.4.2
 */

public class NonSelf {

    // Class variables

    private static Connection dBConnection = null;
    private static PreparedStatement theStatement = null;

    // Instance variables

    private int nonSelfId;
    private int selfHdrId;
    private int clusterHdrId;
    private int scenarioId;
    private int trialId;
    private int realValue;
    private String binaryString;

    /** Constructor to initialize class.
     */

    public NonSelf() {
        nonSelfId = 0;
        trialId = 0;
        scenarioId = 0;
        clusterHdrId = 0;
        selfHdrId = 0;
        realValue = 0;
        binaryString = "";
        return;
    }

    /** Constructor to initialize class.
     */

    public NonSelf(
            final int nonSelfIdIn,
            final int trialIdIn,
            final int scenarioIdIn,
            final int clusterHdrIdIn,
            final int selfHdrIdIn,
            final int realValueIn,
            final String binaryStringIn
            ) {
        nonSelfId = nonSelfIdIn;
        trialId = trialIdIn;
```

```
    scenarioId = scenarioIdIn;
    clusterHdrId = clusterHdrIdIn;
    selfHdrId = selfHdrIdIn;
    realValue = realValueIn;
    binaryString = binaryStringIn;
    return;
}

/** Method to set dBConnection
 */

/**
 * @param dBConnectionIn Database Connection
 */

public static void setdBConnection(final Connection dBConnectionIn) {
    dBConnection = dBConnectionIn;

}

/** create NonSelf.
 */

public final void createNonSelf() {

    this.insertNonSelfRow();

}

/** Set nonSelfId
 */

/**
 * @param nonSelfIdIn
 */

private void setNonSelfId(final int nonSelfIdIn) {
    nonSelfId = nonSelfIdIn;
}

/** Get nonSelfId
 */

/**
 * @return nonSelfId
 */

public final int getNonSelfIdId() {
    return nonSelfId;
}

/** Set selfHdrId
 */

/**
 * @param selfHdrIdIn
 */

private void setSelfHdrId(final int selfHdrIdIn) {
    selfHdrId = selfHdrIdIn;
}

/** Get selfHdrId
 */

/**
 * @return selfHdrIdIn
 */

public final int getSelfHdrId() {
    return selfHdrId;
```

```
}

/** Set clusterHdrId
 */

/**
 * @param clusterHdrIdIn
 */

private void setClusterHdrId(final int clusterHdrIdIn) {
    clusterHdrId = clusterHdrIdIn;
}

/** Get clusterHdrId
 */

/**
 * @return clusterHdrId
 */

public final int getClusterHdrId() {
    return clusterHdrId;
}

/** Set scenarioId
 */

/**
 * @param scenarioIdIn
 */

private void setScenarioId(final int scenarioIdIn) {
    scenarioId = scenarioIdIn;
}

/** Get scenarioId
 */

/**
 * @return scenarioId
 */

public final int getScenarioId() {
    return scenarioId;
}

/** Set trialId.
 */

/**
 * @param trialIdIn
 */

private void setTrialId(final int trialIdIn) {
    trialId = trialIdIn;
}

/** Get trialId
 */

/**
 * @return trialId
 */

public final int getTrialId() {
    return trialId;
}

/** Set binaryString
 */
```

```
    /**
     * @param binaryStringIn
     */

    private void setBinaryString(final String binaryStringIn) {
        binaryString = binaryStringIn;
    }

    /** Get binaryString
     */

    /**
     * @return binaryString
     */

    public final String getBinaryString() {
        return binaryString;
    }

    /** Set realValue
     */

    /**
     * @param realValue
     */

    private void setRealValue(final int realValueIn) {
        realValue = realValueIn;
    }

    /** Get realValue
     */

    /**
     * @return realValue
     */

    public final int getRealValue() {
        return realValue;
    }

    /** Method to Insert NonSelf Row.
     */

    private void insertNonSelfRow() {
        try {

            theStatement = dBConnection.prepareStatement(
                    "INSERT INTO nonself"
                            + " (id, trial_scnr_ch_shdr_id, trial_scnr_ch_id,"
                            + " trial_scnr_id, trial_id, binary_string,"
                            + " real_value) VALUES (?, ?, ?, ?, ?, ?, ?)");

            theStatement.setInt(1, this.nonSelfId);
            theStatement.setInt(2, this.selfHdrId);
            theStatement.setInt(3, this.clusterHdrId);
            theStatement.setInt(4, this.scenarioId);
            theStatement.setInt(5, this.trialId);
            theStatement.setString(6, this.binaryString);
            theStatement.setInt(7, this.realValue);

            theStatement.executeUpdate();

            theStatement.close();

        } catch (SQLException sqle) {

            System.err.println(sqle);
        }
    }
}
```

Reference List

Angeline, P. J. (1995). Adaptive and self-adaptive evolutionary computations. *Computation Intelligence: A Dynamic Systems Perspective* pp. 152-163.

Angeline, P. J. (1996). Two self-adaptive crossover operations for genetic programming. *Advances in Genetic Programming 2* pp. 89-100. Cambridge, MA: MIT Press.

Angeline, P. J., & Pollack, J. (1993). Evolutionary module acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*.

Ayara, M., Timmis, de Lemos, R., de Castro, L., & Duncan, R. (2002a). An investigation into negative selection for change detector generation. *Technical Report, August 2002 University of Kent at Canterbury*.

Ayara, Timmis, de Lemos, de Castro, L., & Duncan, R. (2002b). Negative selection: How to generate detectors. In *Proceedings of ICARIS (International Conference on Artificial Immune Systems)* pp. 89-98.

Baba (1981). Convergence of random optimization methods for constrained optimization methods. *Journal of Optimization Theory and Applications, 33,* pp. 451-461.

Bäck, T. (1992). Self-adaptation in genetic algorithms. In F.J. Varela and P. Bourgine, *Proceedings of the 1st European conference on artificial life* pp. 263-271. Cambridge, MA: MIT Press.

Bäck, T., Fogel, D., & Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. Institute of Physics Publishing.

Bäck, T., Hoffmeister, F., & Schwefel, H. P. (1991). A survey of evolution strategies. *Proceedings of the 4th international conference on genetic algorithms* pp. 2-9. San Mateo, CA.

Bäck, T., Rudolph, G., & Schwefel, H. P. (1993). Evolutionary programming and evolution strategies: similarities and differences. In Fogel and Atmar, *Proceedings of the second annual conference on evolutionary programming* (241), pp. 11-22. San Diego, CA: Evolutionary Programming Society.

Beasley, D., Bull, D. R., & Martin, R. R. (1993). An overview of genetic algorithms: Part 1, Fundamentals. *University Computing*, 15, (2), pp. 58-69.

Beyer, H.-G. (1993). Toward a theory of evolution strategies: some asymptotical results from the $(1 + \lambda)$ theory. *Evolutionary Computation, 1,* (2), pp. 165-188.

Beyer, H.-G. (1995a). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation, 3*, pp. 311-347.

Beyer, H.-G. (1995b). Toward a theory of evolution strategies: the (μ, λ) - theory. *Evolutionary Computation, 2,* (4), pp. 381-407.

Beyer, H.-G. (1995c). Toward a theory of evolution strategies: on the benefit of sex the (μ/ μ, λ) - theory. *Evolutionary Computation, 3,* (1), pp. 81-111.

Beyer, H.-G. (1999). On the analysis of self-adaptive evolutionary algorithms. *Technical Report No. CI-69/99*.

Beyer, H.-G., Schwefel, H.-P., & Wegener, I. (2002). How to analyze evolutionary algorithms. *Technical Report No. CI-139/02* University of Dortmund.

Bremermann, H.J. (1962). Optimization through evolution and recombination. *Proceedings Conference on Self-organizing Systems, eds*. M.C. Yovits, G.T. Jacobi, and G.D. Goldstine, pp. 93-106, Spartan Books

Chellapilla, K., & Fogel, D. B. (1997). Exploring self-adaptive methods to improve the efficiency of generating approximate solutions to traveling salesman problems using evolutionary programming. *Proceedings of the 6th International Conference on Evolutionary Programming* pp. 361-371. Indianapolis, Indiana: Springer.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection* (1st). Retrieved June 23, 2003 from http://www.esp.org/books/darwin/origin/facsimile/title3.html.

Dasgupta, D. (1998). *Artificial Immune Systems and Their Applications*. Springer.

Dasgupta, D., & Attoh-Okine, N. (1997). Immunity-based systems: A survey. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*.

Dasgupta, D., & Forrest, S. (1999). Artificial immune systems in industrial applications. In *Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials (IPMM)*.

Dasgupta, D., & Gonzalez, F. (2002). An immunity based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation* pp. 1081-1088.

Dasgupta, D., & Majumdar, N. S. (2002). Anomaly detection in multidimensional data using negative selection algorithm. In *Proceedings of the Congress on Evolutionary Computation at WCCI* pp. 1039-1044.

de Castro, L., & Timmis, J. (2002a). *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer.

de Castro, L., & Timmis, J. (2002b). An artificial immune network for multimodal function optimization. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC'02), 1*, pp. 699-674.

de Castro, L., & Von Zuben, F. J. (2000a). Artificial immune systems: Part II - A survey of applications. *Technical Report DCA-RT 02/00*.

de Castro, L., & Von Zuben, F. J. (2000b). The clonal selection algorithm with engineering applications. In *Proceedings of GECCO'00, Workshop on Artificial Immune Systems and Their Applications* pp. 36-37.

de Castro, L., & Von Zuben, F. J. (2002c). Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems, 6*, (3), pp. 239-251.

D'haeseleer, P. (1996). An immunological approach to change detection: Theoretical results. *9th IEEE Computer Security Foundations Workshop* pp. 110-119.

D'haeseleer, P., Forrest, S., & Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. *IEEE Symposium on Security and Privacy*.

Droste, S., Jansen, T., & Wegener, I. (2000). Dynamic parameter control in simple evolutionary algorithms. In *Foundations of Genetic Algorithms (FOGA 2000)*, pp. 275-294.

Eiben, A., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3, pp. 124-141.

Esponda, F., Forrest, S., & Helman, P. (2002). A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man and Cybernetics*, 34, (1), pp. 357-373

Fitzpatrick, J., Secrist, J., & Wright, D. J. (1998). *Secrets for a successful dissertation*. Sage Publications.

Fogel, D. B. (1992). An analysis of evolutionary programming. In D.B. Fogel and W. Atmar, *Proceedings of the first annual conference on evolutionary programming* pp. 43-51. La Jolla, CA.

Fogel, D. B. (1994). Asymptotic convergence properties of genetic algorithms and evolutionary programming: Analysis and experiments. *Cybernetics and Systems, 25,* (3), pp. 389-407.

Fogel, D. B. (1997). The advantages of evolutionary computation. In *BCEC97*, Springer, pp. 1-11.

Fogel, D. B., & Chellapilla, K. (1998). Revisiting evolutionary programming. *Aerosense'98: Aerospace / Defense Sensing and Controls*.

Fogel, L. J., Angeline, P., & Fogel, D. B. (1995). An evolutionary programming approach to self-adaptation on finite state machines. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA, pp. 335-365

Fogel, L., Owens, A. J., & Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution* New York: John Wiley.

Forrest, S., & Hofmeyr, S. (2001). Immunology as information processing. *Design Principles for the Immune System and Other Distributed Autonomous Systems,* edited by L.A. Segel and I. Cohen. Santa Fe Institute Studies in the Sciences of Complexity. New York: Oxford University Press, pp. 361-387.

Forrest, S., & Longstaff, T. A. (1996). A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* pp. 120-128.

Forrest, S., Hofmeyr, S. A., & Somayi, A. (1997). Computer immunology. *Communications of the ACM, 40*, (10), pp. 88-96.

Forrest, S., Perelson, A. S., Allen, L., & Cherukuri, R. (1994). Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy* pp. 202-212.

Forrest, S., Perelson, A. S., Allen, L., & Cherukuri, R. (1995). A change detection algorithm inspired by the immune system. *IEEE Transactions on Software Engineering*.

Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers. II Effects of linkage or rates of advance under selections. *Australian Journal of Biological Sciences, 10*, pp. 492-499.

Glickman, M., & Sycara, K. (1998). Evolutionary algorithms: Exploring the dynamics of self-adaptation. *Genetic Programming 1998: Proceedings of the Third Annual Conference* pp. 762-769: Morgan Kaufmann.

Gonzalez, F., & Dasgupta, D. (2002). An immunogenetic technique to detect anomalies in network traffic. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)* pp. 1081-1088.

Gonzalez, F., Dasgupta, D., & Kozma, R. (2002). Combining negative selection and classification techniques for anomaly detection. In *Proceedings of the 2002 Congress on Evolutionary Computation* pp. 705-710.

Hart, W. (1996). A stationary point convergence theory for evolutionary algorithms. *Foundations of Genetic Algorithms, 4,* pp. 325-342. San Francisco: Morgan Kauffman.

Hinterding, R. (1995). Representation and self-adaption in genetic algorithms. In *Proceedings of the First Korea-Australia Joint Workshop on Evolutionary Computation.*

Hinterding, R., Michalewicz, Z., & Eiben, A. (1997). Adaptation in evolutionary computation: A survey. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation* pp. 65-69.

Hinterding, R., Michalewicz, Z., & Peachey, T. C. (1996). Self-adaptive genetic algorithm for numeric functions. *Parallel Problem Solving from Nature* pp. 420-429. Springer.

Hofmeyr, S. (1999). *An Immunological Model of Distributed Detection and its Application to Computer Security*. PHD thesis, Department of Computer Sciences, University of New Mexico.

Hofmeyr, S., & Forrest, S. (1999). Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan-Kaufmann, San Francisco, CA* pp. 1289-1296.

Hofmeyr, S., & Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary Computation, 8*, (4), pp. 443-473.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems* Ann Arbor, MI: University of Michigan Press.

Hussain, T. S. (1998). An Introduction to Evolutionary Computation. *1998 CITO Researcher Retreat.*

Institute for Visualization and Perception Research (2001). *Information Exploration Shootout* (http://ivpr.cs.uml.edu/shootout/network.html)

Jansen, T., & Wegener, I. (1999). On the analysis of evolutionary algorithms - A proof that crossover really can help. *Proceedings of the 7th Annual European Symposium on Algorithms* pp. 184-193.

Johnson, N. L., & Kotz, S. (1970). Distributions in Statistics: Continuous Distributions - 2. In Houghton Miflin, Boston.

Kim, J., & Bentley, P. (1999). Negative selection and niching by an artificial immune system. *Genetic and Evolutionary Computation Conference (GECCO'99).*

Kim, J., & Bentley, P. J. (2001). Evaluating negative selection in an artificial immune system for network intrusion detection. *Genetic and Evolutionary Computation Conference 2001 (GECCO - 2001), San Francisco* pp. 1330-1337.

Kim, J., & Bentley, P. J. (2002a). Towards an artificial immune system for network intrusion detection: an investigation of dynamic clonal selection. In *Proceedings of 2002 Congress on Evolutionary Computation* pp. 1015-1020.

Kim, J., & Bentley, P. J. (2002b). Immune memory in the dynamic clonal selection algorithm. In *Proceedings of the First International Conference on Artificial Immune Systems (ICARIS)* pp. 57-65.

Kim, J., & Bentley, P. J. (2002c). A model of gene library evolution in the dynamic clonal selection algorithm. In *Proceedings of the First International Conference on Artificial Immune Systems (ICARIS), Canterbury* pp. 175-182.

Koza, J. R. (1992). *Genetic Programming: In the Programming of Computers by Mean of Natural Selection* Cambridge, MA: MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs* Cambridge, MA: MIT Press.

Krieger, M. J. (1996). *Means and Probabilities: Using Statistics in Science Projects (Experimental Science).* Franklin Watts.

Liang, K. H., Yao, X., & Newton, C. (1998). Dynamic control of adaptive parameters in evolutionary programming. In *Proceedings of the second Asia Pacific Conference on Simulated Evolution and Learning* pp. 42-49.

Mauch, J., & Park, N. (2003). *Guide to the successful thesis and dissertation: A Handbook for students and faculty* (5th). Marcel Dekker.

Michalewicz, Z., & Fogel, D. B. (2000). *How to Solve It: Modern Heuristics* (1st). Springer.

Mischiatti, M., & Neri, F. (2000). Applying local search and genetic evolution in concept learning systems to detect intrusion in computer networks. In *Proceedings European Conference on Machine Learning ECML-2000* Springer-Verlag.

Okhura, K., Matsumura, Y., & Ueda, K. (1999). Robust evolution strategies. *Lecture Notes in Computer Sciences* pp. 10 - 17.

Oprea, M. L. (1999). *Antibody Repertoires and Pathogen Recognition: The Role of Germline Diversity and Somatic Hypermutation*. Ph.D. Thesis, Department of Computer Sciences, University of New Mexico.

Percus, J. K., Percus, O. E., & Perelson, A.S. (1993). *Predicting the Size of T-Cell Receptor and Antibody Combining Region from Consideration of Efficient Self-Nonself Discrimination*. National Academy of Science, 90, pp. 1691-1695.

Pintér (1984). Convergence properties of stochastic optimization procedures. *Math. Operat. Stat., 15,* pp. 405-427.

Porras, P. A., & Valdes, A. (1998). Live traffic analysis of TCP/IP gateways. In *Proceedings of ISOC Symposium of Network and Distributed System Security*. (http://www.sdl.sri.com/projects/emerald/live-traffic.html)

Quagliarella, & Vicini, A. (1999). A genetic algorithm with adaptable parameters. *IEEE Systems' Man and Cybernetics'99 Congress*.

Rappl, G. (1984). *Konvergenzraten von Random Search Verfahren zur globalen Optimierung.* HSBw München, Germany.

Rappl, G. (1989). On linear convergence of a class of random search algorithms. *Zeitschrift f. angew. Math. Mech. (ZAMM) 69* (1), pp. 37 - 45.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution* Stuttgart: Frommann-Holzboog Verlag.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks, 5,* (1), pp. 96-101.

Rudolph, G. (1996). Convergence of evolutionary algorithms in general search spaces. *Proceedings of the third IEEE conference on Evolutionary Computation* pp. 50-54. IEEE Press.

Rudolph, G. (1997). Convergence rates of evolutionary algorithms for a class of convex objective functions. *Control and Cybernetics, 26,* (3), pp. 375-390.

Rudolph, G. (1999). Self-adaptation and global convergence: A counter example. *Proceedings of the Congress on Evolutionary Computation* pp. 646-651. Piscataway, NJ: IEEE Press.

Runarsson, T. P. (2002). Reducing random fluctuations in mutative self-adaptation. *Parallel Problem Solving from Nature (PPSN VII), Lecture notes in computer science* pp. 194-203. Springer-Verlag.

Schoenauer, M., & Michalewicz, Z. (1997). Evolutionary computation. *Control and Cybernetics, 3*, pp. 307-338.

Schwefel, H. P., & Rudolph, G. (1995). Contemporary Evolution Strategies. In F. Morana et al., *Advances in Artificial Life* pp. 893-907. Springer.

Singh, S. (2002). Anomaly detection using negative selection based on the *r*-contiguous matching rule. In *Proceedings of ICARIS 2002*.

Solis, F., & Wets, R. J. (1981). Minimization by random search techniques. *Math. Operations Research, 6,* pp. 19-30.

Somayaji, A., Hofmeyr, S., & Forrest, S. (1997). Principles of a computer immune system. *New Security Paradigms Workshop*. ACM Press.

Sompayrac, L. (1999). *How The Immune System Works*. Blackwell Science.

Spears, W. M. (1995). Adapting crossover in evolutionary algorithms. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pp. 367-384. MIT Press.

Timmis, de Lemos, R., Ayara, & Duncan, R. (2002). Toward immune inspired fault tolerance in embedded systems. In *Proceedings of 9th International Conference on Neural Information Processing* pp. 1459-1463.

Vargas, P. A., de Castro, L., & Von Zuben, F.J. (2002). Artificial immune systems as complex adaptive systems. In *Proceedings of ICARIS (International Conference on Artificial Immune Systems)* pp. 115-123.

Verwoerd, T., & Hung, R. (2002). Intrusion detection techniques and approaches. *Computer Communications, 25*.

Watkins, A., & Boggess, L. (2002). A new classifier based on resource limited artificial immune systems. In *Proceedings of 2002 Congress on Evolutionary Computation, Part of the 2002 IEEE World Congress on Computational Intelligence held in Honolulu, HI, USA, May 12-17, 2002* pp. 1546-1551.

Wegener, I. (2001). Theoretical aspects of evolutionary algorithms. In *Proceedings of the Twenty-Eight International Colloquium on Automata, Languages, and Programming*

Wierzchon, S. (2000). Generating optimal repertoire of antibody strings in an artificial immune system. *Intelligent Information Systems* pp. 119-133. Advances in Soft Computing Series of Physica-Verlag/Springer Verlag.

Wierzchon, S. (2002). Deriving a concise description of non-self patterns in an artificial immune system. *New Learning Paradigm in Soft Computing* pp. 438-458.

Wierzchon, S., & Kuzelewska, U. (2002). Adaptive clusters formation in an artificial immune system. *WAE'02* pp. 131-138.

Yao, X. (1996). An overview of evolutionary computation. *Chinese Journal of Advance Software Research, 3*, (1), pp. 12-29.

Yao, X., & Liu, Y. (1996). Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming*, pp. 451-460, MIT Press, Cambridge, MA.