An Evolutionary Algorithm to Generate Ellipsoid Detectors for Negative Selection

Joseph M. Shapiro

An Evolutionary Algorithm to Generate Ellipsoid Detectors for Negative Selection

THESIS

Joseph M. Shapiro, Civilian

AFIT/GCS/ENG/05-20

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/05-20

An Evolutionary Algorithm to Generate Ellipsoid Detectors for Negative Selection

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Joseph M. Shapiro, B.S.C.S.

Civilian

March 2005

AFIT/GCS/ENG/05-20

An Evolutionary Algorithm to Generate Ellipsoid Detectors for Negative Selection

Joseph M. Shapiro, B.S.C.S.

Civilian

Approved:

/signed/	
Dr. Gary B. Lamont, PhD (Chairman)	date
/signed/	
Dr. Gilbert L. Peterson, PhD (Member)	date
/signed/	
Dr. Robert F. Mills, PhD (Member)	date
/signed/	
Dr. Richard A. Raines, PhD (Member)	date

AFIT/GCS/ENG/05-20

Abstract

Negative selection is a process from the biological immune system that can be applied to two-class (self and nonself) classification problems. Negative selection uses only one class (self) for training, which results in detectors for the other class (nonself). This paradigm is especially useful for problems in which only one class is available for training, such as network intrusion detection. Previous work has investigated hyper-rectangles and hyper-spheres as geometric detectors.

This work proposes ellipsoids as geometric detectors. First, we establish a mathematical model for ellipsoids. We develop an algorithm to generate ellipsoids by training on only one class of data. Ellipsoid mutation operators, an objective function, and a convergence technique are described for the evolutionary algorithm that generates ellipsoid detectors. Testing on several data sets validates this approach by showing that our algorithm generates good ellipsoid detectors. Against artificial data sets, the detectors generated by our algorithm match $> 90\%$ of nonself data with 0% false alarm. Against a subset of data from the 1999 DARPA MIT intrusion detection data, the ellipsoids generated by our algorithm detect $\sim 98\%$ of nonself (intrusions) with a $\sim 0\%$ false alarm rate.

Table of Contents

Page

Abstract iv

List of Figures viii

List of Tables x

List of Symbols xi

List of Abbreviations xiii

List of Algorithms xiv

I. Introduction 1
- 1.1 Network Security and Artificial Immune Systems 1
- 1.2 Problem Statement 2
- 1.3 Approach 3
- 1.4 Document Overview 3

II. Background 5
- 2.1 Symbolic Problem Definition 5
- 2.2 Network Intrusion Detection 6
  - 2.2.1 Features of an Intrusion Detection System 7
  - 2.2.2 Taxonomy of Intrusion Detection System Structure . 7
  - 2.2.3 Signature Detection v. Anomaly Detection 8
  - 2.2.4 Network Anomaly Detection Problem 9
  - 2.2.5 Network Data Models 10
- 2.3 Finding the Malicious Data Points in Network Data 13
  - 2.3.1 Machine Learning and Statistical Techniques for Identifying Nonself 14
- 2.4 Artificial Immune Systems as a Machine Learning Approach 18
  - 2.4.1 Artificial Immune Systems Background 18
- 2.5 Clonal Selection AIS 19
  - 2.5.1 Mapping from BIS to AIS 19
- 2.6 AIS Inspired Algorithms 28
  - 2.6.1 Clonal Selection 31
- 2.7 Network Data Classification 32
- 2.8 Human Interaction in Searching for Good Antibodies 33
- 2.9 Testing Network Intrusion Detection Systems 34
  - 2.9.1 Network Intrusion Data Sets 34
  - 2.9.2 Network Intrusion Detection System Testing Methods 35
- 2.10 Evolutionary Algorithms 36
- 2.11 Summary 36

Page

III. High Level Design 38
3.1 Ellipsoids as Detectors 38
3.2 Ellipsoids 39
3.2.1 Mathematical Definition of Ellipsoid 39
3.2.2 Volume of an Ellipsoid 41
3.2.3 Is a Point Inside of an Ellipsoid? 43
3.3 An Evolutionary Algorithm to Evolve a Set of Ellipsoids . . 44
3.3.1 Crossover With Ellipsoids 45
3.3.2 Mutating an Ellipsoid 46
3.3.3 Objective Function 55
3.3.4 Convergence 68
3.3.5 Algorithm Parameters 70
3.4 Algorithm Summary and Complexity 71
3.5 Implementation Details 71
3.5.1 Feature Representation: Binary v. Real Value 71
3.5.2 Implementation Language 72

IV. Experimental Design 75
4.1 Validation of Model 76
4.1.1 Pedagogical Problems 76
4.1.2 Real World Test Data Sets 81
4.2 Test Data Set Taxonomy 86

V. Results and Analysis 88
5.1 Pedagogical Problems 88
5.2 MIT Lincoln Labs Data 101
5.3 Iris 105

VI. Conclusion 117
6.1 Research Problem 117
6.2 Contributions 118
6.2.1 Pedagogical Data Sets 119
6.2.2 Algorithm to Evolve Ellipsoids 119
6.2.3 2^n-way Tree for Ellipsoid Approximations 119
6.2.4 Ellipsoids as Detectors 119
6.3 Future Work 119
6.4 Summary 121

Page

Appendix A. Biological Immune System Background 122
A.1 Overview 123
A.1.1 Distributed with no Central Control 123
A.1.2 Memory 123
A.1.3 Multi-Layered 124
A.1.4 Preventative 126
A.1.5 Pattern Recognition 126
A.1.6 Positive Selection 126
A.1.7 Negative Selection 127

Appendix B. Random Number Generator 128

Appendix C. Algorithm Pseudocode 129

Bibliography 136

Index 1

Author Index 1

List of Figures

Figure Page

1. Kim's AIS model 25
2. Harmer's AIS Model 26
3. Dasgupta's Variable-Sized Detectors 30
4. Self Membership Levels 31
5. Elllipsoid Variables 40
6. Good Crossover 47
7. Semiaxis Mutation 48
8. Center Mutation Along Coordinate Axis 50
9. Semiaxis Mutation 51
10. Orientation Mutation PDF 52
11. Degenerate Ellipsoid Intersection 58
12. Center Distance Overlap Problems 59
13. Spherical Overlap Check Problems 60
14. Segment Tree 63
15. 2^n-way Tree Approximation Limitation 65
16. PD1 77
17. PD2 78
18. PD3 78
19. PD4 78
20. PD5 79
21. PD6 79
22. Ellipsoid v. Sphere Detector Bar Graph 89
23. Ellipsoid Evolution, PD1 91
24. Ellipsoid Evolution, PD2 92
25. Ellipsoid Evolution, PD3 93

Figure Page

26. Ellipsoid Evolution, PD4 94

27. Ellipsoid Evolution, PD5 95

28. Ellipsoid Evolution, PD4 96

29. Ellipsoid Performance v. Generation, PD1-PD6 97

30. Sphere Performance v. Generation, PD1-PD6 98

31. Coverage v. Detectors, Ellipsoid and Sphere, PD1-PD6 99

32. Final Spheres after Evolution, PD1-PD6 100

33. Ellipsoid Performance v. Generation, MIT Data Set 102

34. EA Performance v. Classification Performance Discrepancy 103

35. Sphere Performance v. Generation 103

36. Performance v. Detector Count, Ellipsoid and Sphere 104

37. Coverage Comparison, Ellipsoid v. Sphere, MIT Data Set 104

38. Classification v. Generation, Ellipsoid and Sphere, Setosa 107

39. Classification v. Generation, Ellipsoid and Sphere, Versicolor 108

40. Classification v. Generation, Ellipsoid and Sphere, Virginica 109

41. Classification v. Detectors, Ellipsoid and Sphere, Setosa 110

42. Classification v. Detectors, Ellipsoid and Sphere, Versicolor 111

43. Classification v. Detectors, Ellipsoid and Sphere, Virginica 112

44. ROC, Ellipsoid and Sphere, Train Setosa 113

45. ROC, Ellipsoid and Sphere, Train Versicolor 114

46. ROC, Ellipsoid and Sphere, Train Virginica 115

List of Tables

Table		Page
1.	Summary of Network Data Models	13
2.	Mapping BIS to AIS	21
3.	Classification Errors	33
4.	Volume of Hyper-Sphere	42
5.	Pedagogical Data Set Parameters	80
6.	Iris Data Set Parameters	83
7.	Normalization of MIT Data Set	84
8.	MIT Data Set Week Two Attack Profile	85
9.	MIT Data Set Parameters	85
10.	Test Data Set Characteristics	87
11.	Performance of Ellipsoids and Spheres, PD1-PD6	89
12.	Negative Selection Algorithm Comparison on MIT Data Set	105
13.	Comparison of Ellipsoid, Sphere, V-Detector on Iris Data Set	116

List of Symbols

Symbol		Page
F	Set of Fields in a Sample	5
c	number of fields in a sample	5
B	its for each field in a network sample	5
b_i	number of bits in field f_i	5
f_i	i^{th} field in F	5
P	All possible points in sample space	5
p	a point in sample space($p \in P$)	5
N	set of nonself points ($N \subseteq P$)	5
S	set of self points	5
D	set of detectors	5
d_i	detector ($d \in D$)	6
$\mathbb{N}$	all networks	9
$\mathbb{T}$	all time periods	9
$\ltimes$	a network ($\ltimes \in \mathbb{N}$)	9
t	time period ($\approx \in \mathbb{T}$)	9
Δ	subset of all points ($\Delta \subseteq P$)	9
s	set of self points	10
n	set of nonself points	10
x_0	center of ellipsoid along x coordinate axis	39
y_0	center of ellipsoid along y coordinate axis	39
a	half length of semiaxis oriented along x coordinate axis in 2d ellipsoid	39
b	half length of semiaxis oriented along y coordinate axis in 2d ellipsoid	39
ω	center of n-d ellipsoid	39
$\mathbf{V}$	orthonormal semiaxis orientations of ellipsoid	39

Symbol		Page
$\mathbf{\Lambda}$	matrix that defines semiaxis lengths	40
ℓ_i	length of i_{th} semiaxis	40
EA	Evolutionary Algorithm	44
v	orthonormal matrix	50

List of Abbreviations

Abbreviation		Page
AIS	Artificial Immune System	1
BIS	Biological Immune System	2
NID	Network Intrusion Detection	2
ID	Intrusion Detection	3
IDS	Intrusion Detection Systems	6

List of Algorithms

Algorithm Page

1. *insertEllipsoid* 129
2. *insertEllipsoid* 130
3. *generateEllipses* 131
4. *growEllipse* 132
5. *mutateCenter* 132
6. *mutateLength* 132
7. *mutateOrientation* 133
8. *insertEllipsoidIntoSegmentTree* 134
9. *allCornersInside* 135
10. *numPointsCoveredInSegmentTree* 135

An Evolutionary Algorithm to Generate Ellipsoid Detectors for Negative Selection

I. *Introduction*

One of the most harmful things that can happen to an organization is to have a malicious network intrusion. This can cause the loss of revenues, data, productive work time, and customer confidence. For government and military entities, a successful intrusion can cause all of the aforementioned problems and result in the compromise of data or information that is important to national security or military might. Needless to say, it is of utmost importance to protect networks from attacks.

1.1 Network Security and Artificial Immune Systems

Network security is the field that deals with protecting networks from malicious activity. Although it is not a new research area [50], network security has recently gained prominence because of the ubiquity of networks, an increase in the frequency of attacks, and an increase in the resulting damage. Many network security solutions have been proposed and implemented. These include virus detection, firewalls, and anomaly detection.

In recent years, several research efforts have noticed striking similarities between the problem of protecting a network from malicious attacks and the way that the body defends itself against harmful invaders such as viruses and bacterial infections. This observation resulted in a new algorithmic paradigm, the artificial immune system (AIS).

Many research efforts have identified negative selection as an AIS process that has many desirable properties in the computational domain. Negative selection generates detectors that are released into the body to seek out harmful invaders. The most impressive characteristic of the negative selection algorithm is that detectors do not misidentify good self inhabitants as harmful invaders. It is also remarkable that such detectors can be generated when they do not have harmful invaders to "train" on.

Negative selection maps smoothly to the computer domain. The good self inhabitants in the biological immune system (BIS) are normal activity or data in a computer. Harmful invaders from the BIS are malicious attacks in the computer domain. We identify normal activity/data in the computer domain as “self”, while malicious entities are labelled “nonself.”

Detectors generated by negative selection can be used to recognize nonself data in a two-class problem that provides only self data for training. Network intrusion detection (NID) is such a problem. There is abundant self data to train on but only minimal nonself data, since intrusions are often new and unique. NID is not the only application for negative selection. Negative selection can be used to generate detectors for any class of any classification problem. Application domains include medical data, factory anomaly detection, plant classification, and more.

1.2 Problem Statement

We propose to extend the negative selection work of Dasgupta [39], Williams [96], and others. These researchers have investigated hyper-geometric detectors including hyper-rectangles and hyper-spheres. The goal is to improve techniques for generation, representation, and properties of nonself detectors by using hyper-ellipses (hereafter referred as ellipsoids), which provide a more flexible geometry than spheres and rectangles. Thus, the objectives of this research are:

- Establish a mathematical model for n-dimensional ellipsoids. This model should provide the flexibility to describe any n-dimensional ellipsoid. Also, this model should provide the ability to manipulate the size, orientation, and location of an ellipsoid.
- Develop/Identify an algorithm to generate a set of detectors by training on a self class from any n-dimensional data set.
- Validate the algorithm by running it against several pedagogical and real world data sets. Performance is quantified by measuring detection rates (correct identification of nonself) and false alarm rates (incorrect detection of nonself). These metrics are compared against results of other research efforts and against different detector types.

1.3 Approach

These objectives are obtained while achieving, as much as possible, the following design metrics for a negative selection algorithm.

- Accurate Classification-the algorithm should be able to classify nonself data based on detectors that it builds during a training phase that uses only self data. Classification accuracy has two components: (1) true positive rate: the fraction of nonself test points correctly classified as nonself and (2) false positive rate: the fraction of self test points incorrectly classified as nonself.
- Efficiency
  - Minimize Detector Generation Time-if it takes too long to generate detectors, the algorithm is not desirable. This is especially true if detectors must be generated often. If detector generation occurs less frequently, this issue is less important.
  - Minimize Detector Size-this applies to both the size of one detector and the total number of detectors. If a set of detectors takes up too much space, its memory requirements may be more costly than received benefits.
  - Minimize Detection Time-the importance of this design goal is application dependent. In some environments, such as NID, it may be necessary to compare incoming network data against detectors at a real-time rate. In other applications, however, time may not be such an issue.

1.4 Document Overview

Chapter II presents a formal definition of the problem, followed by a summary of intrusion detection (ID), AIS, and machine learning. This is followed by a review of research in these areas. Chapter III sets forth a mathematical ellipsoid model and then describes a negative selection algorithm that generates a set of n-dimensional ellipsoids ("nonself" detectors) by training on "self" data. The Experimental Design chapter describes testing goals, a methodology to accomplish those goals, and the data sets used in testing. This is followed by Chapter V, which reports test results and analyzes the results in light of research objectives. The conclusion (Chapter VI) restates the problem and research objectives, summarizes results, and concludes with suggestions for further research.

II. Background

This chapter provides background in areas that are important to this research. First, we define the generic problem with an English description complemented by a mathematical description using symbolic notation. Section 2.2 provides background on the specific NID problem. Section 2.4 introduces AISs and discusses mapping of the BIS to an AIS that can be applied to classification problems. Section 2.6 presents the negative selection algorithm and analyzes it in depth. Finally, Section 2.10 summarizes evolutionary algorithms, which can be used used to generate detectors for a negative selection algorithm.

2.1 Symbolic Problem Definition

This problem assumes that input consists of training data from only one class, the self class. The problem is to generate a set of detectors that have a high affinity for nonself points (match nonself points) and a low affinity for self (do not match self points). Performance of a generated set of detectors is measured using a set of test points. The set of test points does not include any of the same points that are used for training. Also, the test set may contain points from any classes. Any data from a different class than training (self) is classified as nonself. Labelling all points as self or nonself according to these criteria transforms a n-class problem into a two-class (self, nonself) problem.

Symbolically, the problem is formulated as follows: Let $F= \{f_1, f_2, ..., f_c\}$ be the set of c fields in a data sample. Let $B= \{b_1, b_2, ..., b_c\}$ be the number of bits used to describe each field in the sample, such that b_i is the number of bits in field f_i. Then field f_i can take on 2^{b_i} different values, namely $\{0, 1, 2, ..., 2^{b_i} - 1\}$. Let $P= \{0, 1, ..., 2^{b_1} - 1\} \times \{0, 1, ..., 2^{b_2} - 1\} \times ... \times \{0, 1, ..., 2^{b_c} - 1\}$ be the set of all possible points in sample space, such that $p\in P$ is an $c-$tuple where the i_{th} value of the tuple is a value for field f_i. Let $N\subseteq P$ be the set of all nonself points and let $S= P - N$ be the set of all self points.

Let $\mathbb{D}$ be all possible detectors. Let $D= \{d_1, d_2, ..., d_{detector\_count}\}$ be a set of $detector\_count$ detectors. Define the function

$$match : \{d|d \text{ is a detector}\} \longrightarrow 2^P, \tag{1}$$

(2^P is all the subsets of P). *match* is the function that takes a detector as an argument and returns all points in P that the detector matches. For each $d_i \in D$, $match(d_i) \subseteq P$. The objective is to cover as much nonself space as possible while covering as little self space as possible. These optimization criteria are formally described as follows:

$$\text{maximize} \quad \left| \left(\bigcup_{d \in D} match(d) \right) \bigcap N \right| \tag{2}$$

$$\text{minimize} \quad \left| \left(\bigcup_{d \in D} match(d) \right) \bigcap S \right| \tag{3}$$

2.2 Network Intrusion Detection

ID has been studied for many years [26], although the amount of research being done in this area has increased in recent years because of the ubiquity and risk of the threat. Almost every computer is connected to some sort of network and any damage (by means of an intrusion) can cause at least some inconvenience, as well as lost money and work. For a detailed discussion of intrusion detection, see the work of Amoroso [14], Escamilla [30], Peikari [74]. This section provides definitions to understand ID, a taxonomy of (IDS), and a problem definition for the ID problem.

First, we provide several definitions of intrusion detection:

- **intrusion:** Mukherjee [69] defined an intrusion by its results: loss of confidentiality, denial of resources, loss of integrity, or unauthorized use of resources.
- **intrusion detection:** Amoroso defined intrusion detection as follows [14]: "Intrusion detection is the process of identifying and responding to malicious activity targeted at computing and networking resources."
- **intrusion detection system:** Kim et al. [55] defined a intrusion detection system as "an automated system for the detection of computer system intrusions." Network intrusion detection systems are built on the assumption the behavior of an intruder is significantly different than the behavior of a legitimate user [69].

2.2.1 Features of an Intrusion Detection System. Kim et al. enumerated seven features that a good network intrusion detection system must have [55]:

- **Robustness:** it should have multiple detection points, so that it is not easily subverted by intruders.
- **Configurability:** it should be able to configure itself to the local requirements of each heterogeneous host or network component.
- **Extendibility:** it should be easy to extend network monitoring to newly added network components.
- **Scalability:** as the volume of the data to be analyzed grows, the system should be able to handle the increased work without significant system or network performance degradation.
- **Adaptability:** it should dynamically adapt to detect network intrusions that change dynamically.
- **Global Analysis:** it should be able to correlate events from different hosts that may collectively constitute an attack.
- **Efficiency:** it should be lightweight enough that the resulting system and network performance degradation from running the intrusion detection system is minimal.

Of course, it is difficult build a system that fully satisfies all of these features. However, many systems have been constructed. The next section presents a classification of IDSs.

2.2.2 Taxonomy of Intrusion Detection System Structure. Current network IDSs can be grouped into three categories: monolithic, hierarchical, and cooperative [54].

- **Monolithic:** The network intrusion analysis all takes place on one central machine. All host machines communicate raw data to the central machine, which then analyzes the data. Some of the serious drawbacks of this approach are that it is not scalable and not robust. It is not scalable because adding hosts to the network causes a communication burden as additional raw data is passed to the central processing host. Additional hosts also result in computational burden on the central processing host because it must process data from more hosts. As the number of hosts increases, the performance degradation of the network and the central processing machine is significant. This approach is not robust because the loss of the central processing node (resulting from a successful attack, malfunction, etc.) disables the entire intrusion detection system.

- **Hierarchical:** This approach was developed to overcome the scalability drawbacks of the monolithic approach. In the hierarchical approach, network hosts analyze their own data and then send the results to a central machine. Because each host analyzes its own data, it sends less information to the central machine. As a result, the central machine has significantly less processing to do since the data has already been locally processed at its originating host. Although this approach addresses the scalability issue, robustness is still an issue because the loss of the central node still disables the entire network IDS.
- **Cooperative:** The cooperative approach addresses the robustness issue by distributing all central processing tasks to the individual hosts. The hosts communicate to accomplish coordination and collaborative intrusion detection efforts. The only drawback to this method is that the communication between hosts to accomplish cooperative intrusion detection may be significant, which reintroduces the issue of network scalability.

IDSs of all three types have been implemented, each with the noted advantages and disadvantages. Once the structure question has been resolved, an IDS designer must decide how to detect intrusions.

2.2.3 Signature Detection v. Anomaly Detection. There are generally two types of intrusion detection: signature detection and anomaly detection. Signature detection is comparing known attacks against new network data. A match signifies that some virus or malicious entity has been encountered.

Anomaly detection involves building a model of "normal" behavior and then analyzing the network. When some behavior occurs that deviates from the model of normal behavior by more than some threshold, then that behavior is labelled as an anomaly. It may be immediately flagged as having malicious intent, or simply that it is deserving of further analysis.

The field of signature detection is more established than anomaly detection. This is because signature detection is a simpler problem. Companies such as Symantec [8], Panda [7], and McAfee [5] market proven, industrial strength antivirus products. The shortcoming of signature detection is that it does not protect a computer system from a virus unless the virus is in a virus signature definition database. This database receives periodic updates with new virus signatures so that they can detect all common viruses. Because signature detection is a well studied area with solid commercial products available, we choose rather to focus on anomaly detection. Anomaly detection is a harder problem. Rather than trying to find matches with previously encountered attacks (signature detection), anomaly detection attempts to build a model that essentially predicts what is an attack and what is not. The following section describes the anomaly detection problem.

2.2.4 Network Anomaly Detection Problem. In the following discussion, the word "anomalous" is synonymous with "malicious". The problem is to build a function whose input is some network data and whose output is exactly those data points in the input that are anomalous.

Symbolically, the problem is defined as follows: Let $\mathbb{N}$ bet the set of all networks. Let $\mathbb{T}$ be the set of all possible time periods. Let *toData* be a function that maps the behavior of a network over some time period(s) to a set of data points.

$$toData : \mathbb{N} \times 2^{\mathbb{T}} \longrightarrow 2^{P}, \tag{4}$$

where $2^{\mathbb{T}}$ is the set of all subsets of $\mathbb{T}$ and 2^{P} is the set of all subsets of P, where P retains its definition from Section 2.1.

Given a network $\ltimes \in \mathbb{N}$ and a time period $t \in \mathbb{T}$, *toData* maps $(\ltimes, \approx)$ to $\Delta \subseteq P$. That is,

$$toData(\ltimes, \approx) = \Delta \tag{5}$$

The normal (self) and malicious (nonself) portions of Δ are defined as follows. Let $s_\Delta \subseteq \Delta$ be the set of self network data points in Δ and let $n_\Delta \subseteq \Delta$ be the set of malicious network data points in Δ. We note that s_Δ and n_Δ abide by the following constraints:

$$s_\Delta \bigcup n_\Delta = \Delta \tag{6}$$

$$s_\Delta \bigcap n_\Delta = \emptyset \tag{7}$$

Let $findMalicious$ be a function that maps a set of network data points (produced by *toData*) to a set of network data points:

$$findMalicious : 2^P \longrightarrow 2^P \tag{8}$$

The goal is to define $findMalicious$ so that it accepts as input the set of data points for a network over a time period(s) and produces as output exactly the set of malicious data points in the input set. Symbolically, we want to find to define $findMalicious$ such that $\forall \Delta (findMalicious(\Delta) = n_\Delta)$.

This section introduces two functions, *toData* and $findMalicious$, but does not define them. The following sections address the definition of *doData* and $findMalicious$.

2.2.5 Network Data Models. This section addresses the definition of the function *toData*, described in Equation 5. The output of *toData* is usually some set of network features, either raw or processed. Khoshgoftaar et al. divided the attributes from network data into three groups: basic attributes, content Attributes, and traffic Attributes [53]. Basic attributes are the basic features of a network packet (e.g. source IP address, destination IP address, etc. in an IP packet). Content attributes are metrics that summarize the behavior of a network based on the contents of packets. e.g. a number of temporally close attempts to log in with the wrong password may indicate malicious intent. Traffic attributes measure rates of occurrence of network events. These are expensive to compute but often significant in recognizing some types of attacks.

Amoroso [14] listed some techniques to model behavior and look for intrusions: audit train processing, on-the-fly processing, profiles of normal behavior, signatures of abnormal behavior, parameter pattern matching. Of course, there are many ways to model the behavior of a network. The following paragraphs review network data models that have been used in other research efforts.

Payload Byte Value Distribution. Wang and Stolfo [93] analyzed the byte frequency distribution in the payload data of network packets. They constructed a histogram to represent the frequency of each of the 256 possible values of a byte. They built a profile distribution using intrusion-free training data. During detection phase they compared the distribution of the test data to the profile distribution from the training data. When the Mahalanobis distance [29, p.36] between the training and test distributions exceeded a threshold, a warning alert was generated. This novel approach introduced a new type of protection: In order to sneak an intrusion in the payload of a network packet, an attacker must somehow spoof the byte frequency distribution of the data. This is difficult unless the attacker has some knowledge about the normal distribution of the byte values. Getting this knowledge is, of course, difficult without privileged access to the network or host under attack.

System Calls. When any program runs, it makes a series of system calls. Forrest et al. [33] implemented an AIS that uses sequences of UNIX system calls to model self and nonself points. They found that this is a viable approach, as they were able to reliably identify several UNIX intrusions while avoiding false alarms. Hofmeyr [47] asserts that short sequences of system calls executed by running processes are a good discriminator between normal and abnormal operating characteristics in many programs. Hofmeyr noticed that programs tend to execute the same sequences of system calls and that intrusions often follow different paths of system call execution. He found that sequences of length 10 were a good discriminator for the UNIX programs `sendmail`, `lpr`, and `ftpd`.

Upon completion of his PhD, Hofmeyr extended this application to industry. He started a security company called Sana Security [78] [66] that markets an intrusion detection product called Primary Response. Primary Response is based on the idea that sequences of system calls are a good discriminator between good and malicious programs. Sana Security claims that their product "learns normal application behavior by observing code paths in running programs" and then stops anomalous code paths by blocking system call execution [79].

There are many ways to model traces of system calls. Warrender et al. [94] suggested the following models: Simple enumeration of observed sequences, comparison of relative frequencies of different sequences, a rule induction technique, and Hidden Markov Models. Warrender concluded that, for the problem of intrusion detection, weaker methods than Hidden Markov Models are likely sufficient. They hypothesized that weaker methods were sufficient because the system call data are regular enough for even simple modelling methods to work well.

Network Packet Header. Every network packet has a header with several fields. Williams [97] and Harmer [44] used the fields in TCP, IP, ICMP, and UDP packets to model network behavior. If there are n fields in a header, this technique simply maps a network behavior to a point in n-dimensional space.

Network Statistics. Gonzalez and Dasgupta [41] used network statistics including bytes per second, packets per second, and ICMP per second. Mills et al. [67] used the same statistics. Both of these research efforts found that this model of network data viable for identification of some network attacks.

User Shell Command Input. Lane and Brodley [59] used user shell command inputs to model the self nonself space. They mapped the shell command inputs to a class of machine learning known as instance based learning (IBL). One common IBL system is to use k-nearest neighbor. Each new instance is then assigned to the class of the majority of its k nearest neighbors. In their implementation, Lane and Brodley concluded that their technique works well when the data supports sufficient class separation. They also concluded that their system generally detects anomalous conditions quickly while generating false alarms rarely.

Table 1: Summary of Network Data Models

Model	**Research Efforts**
Payload Byte Value Distribution	Wang and Stolfo [93]
System Calls	Forrest et al. [33], Hofmeyr [47] [78] [66] [79], Warrender et al. [94]
Network Packet Header	Williams [97], Harmer [44]
Network Statistics	Gonzalez and Dasgupta [41], Mills et al. [67]
User Shell Command Input	Lane and Brodley [59]
49 bit Representation for TCP SYN packet	Balthrop et al. [16]

Other Definitions for *toData.* Balthrop et al. [16] explored the effects of using a 49 bit representation for a TCP SYN packet. They also explored the results of using a representation mask that allowed for pattern matching in different bit-orders. These masks allowed one nonself detector to detect more points because it detected all points that matched it after permuting their bits according to some mask.

There is an innumerable number of ways to model network data. This is still a current area of research, as the proper model may provide the key to a good anomaly detection system. Table 1 summarizes the data models in the preceding discussion.

2.3 Finding the Malicious Data Points in Network Data

Once an appropriate *toData* function (see Equation 5 has been chosen, the definition of $findMalicious$ (see Equation 8) is at the heart of the intrusion detection problem. The objective of $findMalicious$ is to identify all malicious data points in some input set of data points.

In computer science, the definition of $findMalicious$ is a machine learning problem. That is, we are given a set of n features that describe one datum in the network data, where n is the dimensionality of the elements in the output of the *toData* function. Using these n features, a system is built to classify data points. Since network attacks (nonself) are assumed to be rare relative to the frequency of normal (self) network behavior, the word "anomaly" is sometimes substituted for "attack." This section begins with review of classical machine learning techniques for anomaly detection. Following the classical machine learning techniques, we introduce AIS as a machine learning technique for anomaly detection.

2.3.1 Machine Learning and Statistical Techniques for Identifying Nonself. Machine learning is a field of computer science in which computers programs build a model to learn to classify or group data points. Machine learning is applicable to the network anomaly detection problem because the goal is to classify network data points as being normal (self) or anomalous (nonself). In a network environment, there is often ample normal data to train on. However, since malicious attacks are very diverse and rather rare relative to the frequency of normal data, there is often a limited or non-existant supply of nonself data to train on. Another reason for the short supply of nonself training data is that it is difficult to know what is self and nonself when collecting training data in a real world environment. It is easy, however, to lock down a network and then collect self training data. Another source of self data is to log network activity and then, if a postereori analysis shows that it probably does not contain nonself data, label the data as self.

When training data has been obtained, a machine learning technique can be used to train a model. The usefulness of the model can be tested by measuring classification accuracy on test data. Some machine learning techniques do not scale well to higher dimensional data. For this reason, feature subset selection is sometimes used as a preprocessing step to minimize the dimensionality of the data. For this reason, we include a short discussion of feature subset selection.

Feature Subset Selection. Part of the problem of modelling self and nonself spaces in the network intrusion detection problem is dimensionality. If the selected model has a significant number of features, dimensionality precludes efficiency in classification operations. Pruning features can have a significant impact on the performance of a pattern classification problem, such as differentiating self from nonself. Feature subset selection has been an area of research for as long as classification has been an area of interest. The objective of feature subset selection is to identify the smallest group of features that can be used to build a model that discriminates between different classes well. Thus, as often occurs, there are two competing objectives: (1) discriminate between classes and (2) use the smallest number of features as possible.

Depending on the dimensionality of the model, feature subset selection is important in NID. If the model is the fields in a network packet, then dimensionality is very large if features are not pruned. A TCP packet header has twelve fields in the packet header. High dimensionality occurs often in network data models because they reflect complex, real world problems. Much research has been done on feature subset selection techniques. Marmelstein [64], Peterson [75], and Duda et al. provide discussions on feature subset selection.

After a feature subset has been selected, a learning technique must be chosen to generate a model that can be used for testing on new data. The following discussion reviews machine learning techniques that have been used with NID data.

Unsupervised Learning through Clustering. Zhong et al. [99] suggested that unsupervised learning techniques may be best for NID because temporal changes in the state of a network, as well as in network intrusions, can make classification-based techniques ineffective. They investigated clustering as an unsupervised learning technique in network intrusion detection. In their work, four clustering techniques are considered: K-means, Mixture-Of Spherical Gaussians, Self-Organizing Map, and Neural-GAs. After clustering, Zhong needed a way of labelling clusters as self or nonself. A simple test was chosen and implemented for this purpose. Zhong chose a threshold η. The η^{th} largest clusters were assumed to be self and all other clusters were assumed to be nonself. This labelling technique relies on the assumption that there are more self points than nonself points in all situations. Hence, it is not very robust. In their research, Zhong et al. even found that, in one situation, the second largest of 200 clusters was a nonself attack. The benefit of this technique is that it may be more robust to changing network environments (as Zhong claims but does not support). The detriment, however, is the assumption that large clusters are self and small clusters are nonself.

Detecting Anomalies over Noisy Data. Eskin [31] suggested a technique for detecting anomalies without training on self data. He made the assumption that a data set contains a large number of self elements and relatively few nonself elements. This assumption holds true for the NID domain. This is useful for several reasons: (1) It is often difficult to obtain clean data, (2) training on unclean data has serious side-effects because an intrusion in the training data labelled as clean results in a similar intrusion in the test data being classified as clean and (3) A system that can train on unclean data is adaptive because it can train online, as it works.

To do this, Eskin used learned probability distributions. If there are χ training points, then the algorithm has χ iterations. It assigns each training point to the self set or the nonself set. At the beginning of each iteration, the self and nonself sets are emptied. His algorithm then sequentially analyzes each training point and assigns it to the self or nonself set, depending on the probability distribution that resulted from the previous iteration. At the end of each iteration, a new probability distribution is generated. After the χ^{th} generation, a probability distribution has been created that can be used to classify test points. Eskin applied this technique to NID and found that, in his experiments, the results for noisy data (no clean training data) were similar to published results for other techniques that were allowed to train on clean data. The data Eskin used for his experiments was traces of system calls from the 1999 DARPA Intrusion Detection data set [61].

Correlation. Ning et al. [72] [71] [70] investigated an approach in which network events are correlated to identify malicious sequences of events. The reason for this approach is that "component attacks are usually not isolated, but related as different stages of the attacks, with the early ones preparing for the later ones." They used a directed graph in which nodes represent network events and edges represent causal relationships between network events. By keeping only correlated alerts, Ning used the generated graphs to eliminate many of the false alarms that arise in network intrusion detection systems, providing a valuable service to network administrators who must respond to all intrusion alerts. Although Ning's approach is successful at correlating different parts of an attack, it still relies on the IDS to detect portions of an attack so that they can be correlated. Also, novel attacks are not detected using Ning's approach. For these reasons, Ning suggested that correlation be used with other methods.

Nonstationary Models. Mahoney and Chan [68] used the rate of occurrence of network events to classify events as nonself or self. They found that network events tend to occur in bursts separated by long gaps on the temporal scale. In their model, they assumed that the probability of an event occurring was $\frac{1}{d}$, where d was the time since the most recent occurrence of the event. Hence, when an event occurred that had not occurred in a significant amount of time (more than some selected threshold), this event was classified as nonself. They found that the identified anomalies from the Lincoln Labs Intrusion Detection data complemented the identified anomalies from from other published intrusion detection systems, so they suggested that their approach may complement another approach.

Finite State Machines. Anchor et al. suggested using finite state transducers (FST) to detect network attacks [15]. A finite state transducer is the same as a Mealy-type finite state machine [6]. Anchor used several flavors (lexicographic and Pareto-based) of multi-objective evolutionary algorithms to evolve FSTs that could identify attacks using sequences of network packets as input. Anchor concluded that this idea has promise because it provided good initial results, but whether or not this method is efficient for real world problems remains an open question.

Sekar et al. [80] also investigated the effectiveness of a finite state machine to model normal and anomalous computer behavior. However, they used state machines to model sequences of system calls, an idea introduced by Forrest et al. [33]. They contributed an approach that builds a finite state machine in a computationally and memory efficient manner. They built a state machine for every program, but each state machine took used only a few kilobytes and ran (training and detection) with minimal overhead which was constant for each system call.

Some of these ideas have been implemented in available products. In addition to commercial products, there is a good number of free IDS products that are available. The

Information Assurance Technology Analysis Center [22] provides a good summary of IDS products.

The previous discussion reviews traditional machine learning approaches applied to the NID problem. However, artificial immune systems is a newer machine learning approach that is currently be investigated. The next section reviews AIS, especially the negative selection algorithm, in detail.

2.4 Artificial Immune Systems as a Machine Learning Approach

An AIS is a machine learning technique because it trains on self data and then attempts to recognize nonself data. The machine learning model that is built during the training phase is a set of nonself detectors. This section defines artificial immune systems, provides background on the biological immune system, which is the inspiration for AIS, and summarizes some of the important AIS algorithms from BIS.

2.4.1 Artificial Immune Systems Background. Much of the background provided here on artificial immune systems is from de Castro and Timmis [20], Kim and Bentley [55], and Dasgupta et al. [23]. Also, a short introduction to the human biological immune system is located in Appendix A.

An artificial immune system is an element of the set of bio-inspired algorithms, which includes evolutionary algorithms, neural networks, swarms, ant colony optimization, and others [83]. This sections begins with a definition of AIS. Following the definition, we provide a short discussion of immune network theory and how the clonal selection network theory can be mapped to an AIS.

Definition of AIS. de Castro and Timmis [20, p.58] define Artificial Immune Systems (AIS): **"Artificial immune systems (AIS) are adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving."**

Immune Network Theory. Immune network theory is the study of how entities in the immune system interact ("network") with each other to produce observed immune system behaviors. In immune network theory, there are generally two different schools of thought, the clonal selection theory and the network theory [45]. A third theory, the danger theory, is presently gaining momentum. A description of these three theories is provided in Appendix A. Most AIS models have adhered to the clonal selection theory, and this work continues in that direction. There is not substantiated reason, however, for choosing one over the other. We include a small sampling of network theory work and danger theory work in AIS. Following these samplings, the clonal selection theory AIS is developed in detail.

In the area of the network theory, Timmis and Neal [91] and de Castro and Von Zuben [21] have both implemented an AIS using the network immune model. Timmis and Neal [91] applied the model to unsupervised machine learning and de Castro and Von Zuben [21] applied the model to the problem of clustering and filtering unlabelled numerical data sets.

Danger theory is young in the AIS community. Little research has been done to investigate how this model could be applied to an AIS, although Aickelin believes that there could be some beneficial applications [12].

2.5 Clonal Selection AIS

The clonal selection network theory assumes that interactions in the immune system occur between antibodies and antigens. The following sections show how to map these interactions into an AIS architecture provide a summary of research that has been done using this theory.

2.5.1 Mapping from BIS to AIS. Forrest et al. [34] were among the first to suggest that self/nonself discrimination in a BIS is analogous to protecting a computer from viruses and other malicious code. Forrest et al. [32] observed that the role of natural immune systems in the body is "analogous to that of computer security systems in computing. Although there are many differences between living organisms and computer systems....the similarities are compelling and could point the way to improved computer security." Somayaji, one of Forrest's students, suggested that there are several features of the BIS that are desirable for a computer immune system [82]. These include distributability, diversity, disposability, adaptability, autonomy, dynamic coverage, anomaly detection, multiple layers, identity via behavior, no trusted components, and imperfect detection.

Since then, others have noticed that nonself discrimination is simply a classification problem in which training data exists for only one class [23]. Nunes de Castro and Timmis [20, p.60] suggest that three parts are necessary for a good mapping from the BIS domain to the AIS domain. These three parts are:

- A representation for the components of the system. Often, the components that are mapped into an AIS are the lymphocytes (B-cells and/or T-cells), antibodies, and the antigens.
- A set of mechanisms to evaluate the interaction of individuals with the environment and each other
- Procedures of adaptation that specify how the behavior of the system varies over time.

These three mappings are discussed in the following sections:

2.5.1.1 Representation of the Components of the System. The components of the BIS that are usually most interesting when mapping into an AIS are the lymphocytes/antibodies and antigens. Further, it is not the lymphocytes and antigens that are interesting, but the lymphocyte receptors and the antigen epitopes. Since receptors and antigens are complements of each other (a lymphocyte matches an antigen if the shape of its receptor is close to the complement of the shape of antigen's epitope), the lymphocyte receptor and complement of the antigen epitope are mapped into the AIS domain. The lymphocyte receptor is ofen referred to as the antibody in AIS.

Table 2 is a reference for mapping BIS terms into AIS terms and vice versa.

2.5.1.2 Evaluation of Interaction of Individuals with the Environment and Each Other. In the clonal selection immune network model, interactions are limited to

Table 2: A mapping of important BIS terms into the AIS domain.

BIS	**AIS**
antibody	nonself detector
antigen	data point
self entity	class of data used for training.
nonself entity	data of classes that are not available for training. Detectors are constructed to match nonself data.

those that occur between antibodies and antigens. In an AIS, this maps to interactions between detectors and data points.

In the BIS, the affinity between an antibody and an antigen is a function of several processes including electrostatic interactions, hydrogen bonding, van der Waals interaction, and others [20, p.62]. In AIS, detector/data point affinity is usually measured by an affinity function,

$$affinity : D \times P \longrightarrow \mathbb{R}, \tag{9}$$

that maps a detector and a data point to a real number. A greater output value signifies higher affinity of the detector for the data point.

There are many possible affinity functions, depending on the representation of the data. Following is a short review of some of the more popular affinity functions for binary representations and real-valued representations.

Binary Matching Functions. There are many binary matching functions. These include r-contiguous, r-chunk, Hamming distance, Rogers and Tanimoto [29, p.541], and others.

r-contiguous. The r-contiguous bits matching rule [34] is that two elements, with the same length, match if and only if at least r contiguous characters are identical. More formally, let $x, y \in \{0,1\}^\ell$ be two separate elements. x and y match iff $\exists(1 \leq i \leq \ell) : \forall(i \leq m \leq i+m-1)(x_m = y_m)$ and $i+m-1 \leq \ell$.

r-chunk. Balthrop et al. [16] introduced r-chunks [84] matching, which is a variant of the r-contiguous bits matching rule. The r chunks matching rule

simply specifies a chunk of a string and a starting point. If r contiguous bits match, beginning with the starting point, then the binary strings are classified as matching. The r-chunks matching rule resolved several issues with r-contiguous bits matching. The principal issue was that there were some detectors that could not be generated, so there were "holes".

Hamming Distance. If two bit strings both have the same length, the Hamming distance is simply the number of positions in the strings that are different [44].

Others. Harmer et al. [44] provides a good review of the Rogers and Tanimoto matching function, as well as several other binary matching rules.

Real Valued Matching Functions. Many of the real valued matching functions are intuitive, since they are based on Euclidean space. The following discussions provide a description of Euclidean distance, Manhattan distance, interval constraints, and Mahalanobis distance. For additional discussions of Real valued distance operators, see [42], [29, p.188], [44], [20, p.64-5,70-1].

Euclidean Distance. Whenever the data point space is modeled using real numbers, the most obvious matching function is Euclidean distance [29, p.36] [39] [25] [40]. To test if a point matches some region or point, the Euclidean distance to that region or point is calculated. If the distance is below some threshold, then it is a match. Otherwise, it does not match. The Euclidean distance is a simple and straightforward calculation, as well being easy to conceptualize. In a multi-dimensional space, using Euclidean distance as a threshold to decide whether or not point β matches point α is the equivalent to establishing whether or not β resides inside of some hypersphere h. The hypersphere is defined such that the center is α and the radius is some threshold, t. In an n-dimensional space, the Euclidean distance between points $\mathbf{x} = (x_1, x_2, ..., x_n)$ and $\mathbf{y} = (y_1, y_2, ..., y_n)$ (subscript i signifies i^{th} dimension) is

$$\text{Euclidean Distance } = \sqrt{(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^T} \tag{10}$$

Manhattan Distance. The Manhattan distance is an approximation to the Euclidean distance. It is simply the sum of the differences in each dimension. The advantage is that it is computationally more efficient than calculating the Euclidean since no squares or square roots are computed. The disadvantage is that it is an approximation, so there is a loss in accuracy compared to Euclidean distance.

$$\text{Manhattan Distance } = \sum_{i=1}^{n} |\mathbf{x}_i - \mathbf{y}_i| \tag{11}$$

Interval Constraints. Another matching rule that has been investigated is to use detectors that are coded as interval constraints. Hou et al. [48] considered TCP packets in a network. They chose three features from the TCP packet header: source IP, destination IP and port number (source port number if the packet was leaving the local network, destination port number if the packet was entering the local network). A detector matched a TCP/IP packet if the three field values for a packet fell within the corresponding intervals in the detector. They found that using intervals for matching resulted in comparable performance to using binary bit strings with some bit matching function. They also concluded that it was more understandable to use interval constraints instead of a binary representation. Interval constraints are the same as checking for membership inside of a hyper-rectangle.

Mahalanobis distance. The Mahalanobis distance [4] is similar to the Euclidean distance, with the addition that it takes into account a measure of variance in each dimension and a measure of correlation between different pairs of dimensions. This generally means that if component a of the distance has higher stastical variance than component b, then the a is weighted less when computing the distance metric. The Euclidean distance defines all equidistant points from a reference point as a hypersphere. The Mahalanobis distance defines all equidistant points from a reference point as a hyper-ellipse. To mathematically describe a n-dimensional hyper-ellipse that is not aligned according to the coordinate axes requires a $n \times n$ covariance matrix. If we let A be the $p \times p$ covariance matrix, then the Mahalanobis distance between two points $\mathbf{x} = (x_1, x_2, ..., x_p)$ and

$\mathbf{y} = (y_1, y_2, ..., y_p)$ is defined as follows:

$$\text{Mahalanobis distance}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})A^{-1}(\mathbf{x} - \mathbf{y})^T} \tag{12}$$

Most research has implemented the measurement of affinity between a detector and a datum as a function that measures similarity between two instances of the same representation (i.e. two binary strings). This is not, however, analogously correct from a BIS point of view. In the BIS, an antibody has a high affinity for an antigen if they have complementary binding sites. Although this works well in the BIS, it has not been successfully implemented in a practical AIS because of the difficulty of defining a complementary affinity function. Hart and Ross [45] simulated an artificial immune network using complementary affinity functions in a two-dimensional real valued space. They found that results in their simulations were interesting, although they provided no theoretical explanation and did not map complementary affinity to a practical application.

Using functions that measure interactions between different entities, a full full AIS IDS can be designed. Although this research is not to provide a model for a complete AIS intrusion detection system, we provide a few examples of current research in that area. There has been an evolution from early, simple models (see [52]) to current models that are more complex and map more components of the BIS to the AIS. Kim and Bentley [54] proposed a model which emphasizes the idea of a primary IDS and a secondary IDS, as well as the principle of being distributed. Figure 1 shows this model. Harmer et al. proposed a hierarchical AIS model for an IDS that emphasized collaboration between network hosts and memory of good detectors, although the idea of primary and secondary IDS are not explicitly included. A diagram showing Harmer's model is shown in Figure 2. Lamont [58], Forrest [34], Timmis [20] and others have also proposed architectures for the implementation of an AIS in an IDS problem.

Behavior of System over Time. The BIS defends the immune system by performing several important processes. This section describes several of these processes and how they can be mapped in an AIS.

Figure 1: This diagram from [54] shows an AIS model for an IDS proposed by Kim and Bentley. This model emphasizes the primary and secondary IDS.

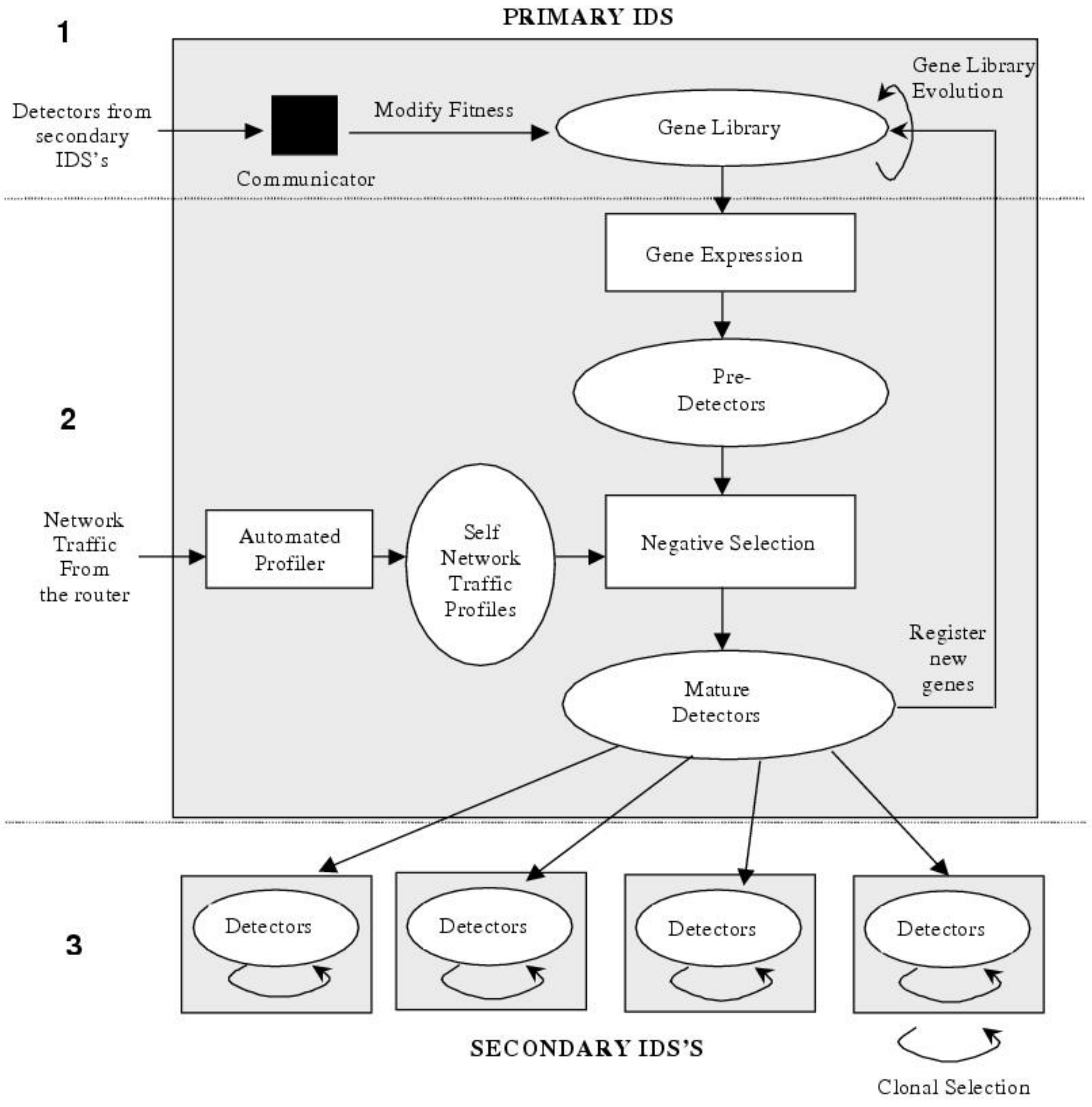
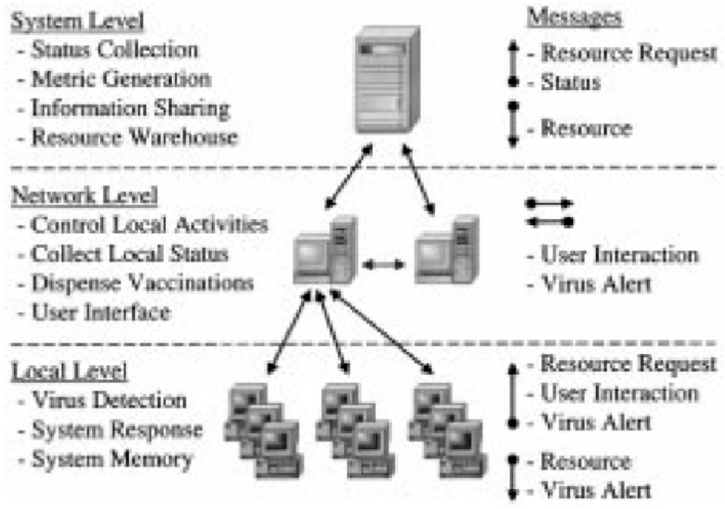


Figure 2: This diagram from Harmer et al. [44] [58] shows an AIS model for an IDS. This model emphasizes the distributed and hierarchical nature of the model.



Generation of New Lymphocytes. The development of new B-cells takes place in the bone marrow. Since the range of receptors (gene sequences) that can be generated is very large, this operation is often mapped to random generation of B-cell receptors in an AIS. However, there are some interesting nuances of B-cell generation that can also be mapped from biology. One of these is gene libraries [60] [92]. The bone does not actually choose genes for receptors randomly. It chooses the genes out of several gene libraries. This can be mapped to an AIS by having gene libraries that are used in the creation of new B-cells. These libraries could include genes that have proven in the past to be good for a particular application or are known to be good through some problem domain knowledge. Little work has been done in the area of using gene libraries for generating new B-cells. This is probably due to the difficulty inherent in obtaining good libraries from which to generate B-cells. From the biological perspective, these good gene libraries evolved over many generations, although these gene libraries do not evolve in individuals, because they are encoded by DNA. With this knowledge, it seems that an evolutionary algorithm could be used to evolve gene libraries, which could then be used in an AIS application to generate new B-cells. Hightower et al. [46] empirically validated the hypothesis that a genetic algorithm can evolve a good gene library, whose fitness is measured by randomly generating antibodies from the gene library and computing their affinity to a set of antigens. Although Hightower et al. showed the potential of evolving gene libraries in a contrived problem, no research efforts have attempted to apply gene library evolution in a practical AIS application.

Positive Selection. In the BIS, T-Cells undergo a process called positive selection in the thymus. Naive T-cells originate in the bone marrow but then migrate to the thymus, where they are tested. For a T-Cell to be effective, it must recognize at least one of the self-MHC molecules or APC (antigen presenting cells) because this is how a T-cell finds infections. If the T-cell does not match at least one self-MHC molecule or APC, then it is usually destroyed through apoptosis. This process is easily mapped to an AIS. Each immature (newly generated) T-cell is compared to the self-MHC molecules and APCs. If it does not, according to a selected affinity function, match at least one self-MHC molecule or APC, then it is destroyed.

Negative Selection. In the BIS, T-cells also undergo a process called negative selection in the thymus. Negative selection destroys those T-cells that recognize self-peptide, i.e. unharmful self molecules. In a computational domain, this process can be mapped into a process which develops detectors for a class of data by interacting only with the complement of that class. This happens by generating T-cells (randomly or with some knowledge) and then checking each T-cell to see if it matches any self points. If a T-cell matches any self points, then it is thrown out. This can be stated as a two-class pattern classification pattern problem in which the classifier gets to train on only one class of points.

Clonal Selection. Clonal selection refers to the behavior of B-cells when they recognize an unwanted antigen [10]. Upon recognition, B-cells are cloned (mitotic cell division). The rate that a B-cell is cloned is directly proportional to its affinity to an antigen. Also, when it is cloned, a B-cell undergoes somatic hypermutation. The rate of mutation is inversely proportional to the affinity of the B-cell to the antigen that it recognized. This is often mapped to an evolutionary strategy [17, ch. 9], which is an evolutionary algorithm that uses mutation but generally not crossover. The evolutionary strategy does have to be modified, however, so that mutation is inversely proportional to affinity and that more individuals of higher affinity are produced and mutated.

2.6 AIS Inspired Algorithms

Significant research has been conducted that attempts to benefit from the strengths of AIS. This research has focused on pattern matching and intrusion detection [13].

Negative Selection.

Generating Detectors Using Negative Selection. The negative selection algorithm to generate nonself detectors is based on the way T-Cells go through negative selection in the thymus. New detectors are randomly generated and then compared against every self point. If a detector matches at least one self point, then it is thrown out and a replacement is randomly generated. This is not computationally efficient. If there are $|S|$ self points and $|N|$ nonself points and it is desired to generate γ good (does not match any self points) detectors, then the time complexity increases exponentially in $|S|$ nd linearly in γ [27].

Forrest et al. [27] suggested that, while this algorithm does work for arbitrary matching rules, better algorithms may be possible for specific matching rules. They proposed some modified negative selection algorithms that scale linearly, although they have some drawbacks because there are "holes," nonself points that cannot be covered because of the matching functions that are used in the algorithms [27]. Another weak point in this algorithm is that it uses binary representations. It is difficult to extract meaningful domain knowledge from a binary representation to analyze the reasons for an anomaly report [40].

Dasgupta et al. [40] used negative selection to build a classifier to distinguish self and nonself data. Their approach was somewhat unique, however, because they did not use negative selection to create detectors. Rather, they used negative selection to generate artificial anomalies. The NID problem usually has only self data to train on. To create artificial anomalies, Dasgupta randomly generated points in the self-nonself space. If the point matched any self points, it was discarded. Otherwise, it was put into a pool of artificial anomalies. With samples from both classes, Dasgupta was able to apply well-studied classifier methods.

One of the first proposed approaches for real valued nonself detectors in a Euclidean space is to use hyper-rectangles [97] [96]. This approach has several benefits. First, it is simple, both conceptually and computationally. It is more efficient to compute whether or not a point falls inside of a hyper-rectangle than it is to compute whether or not a point falls in a hyper-sphere. For a hyper-rectangle, only some simple compares are required, whereas in the hyper-sphere approach square and square root operations are required. Secondly, hyper-rectangles can cover an entire space (they geometrically complement each other), whereas hyper-spheres cannot.

Using Euclidean distance as a threshold to decide whether or not detector d matches point p is the equivalent to establishing whether or not p resides inside of some hypersphere whose center points and radius are specified by d. Dasgupta et al. [40] have used this technique in an AIS to build nonself detectors using negative selection. While hyper-spheres can be combined to cover non-spherical spaces, this often causes overlap. Overlap is not desirable because it means that detectors are doing duplicate work.

Dasgupta et al. [49] improved upon the hyper-sphere representation by using V-detectors, or "variable detectors". The detectors were hyper-spheres, but the radius was variable. This was an improvement because larger hyper-spheres could be used to cover large spaces that contained no self points. Also, smaller hyper-spheres could be used to fit smaller areas of nonself space that were tightly bounded by regions of self space.

Gonzalez and Dasgupta extended the idea of classifying points as self or nonself to levels of "selfness" or "nonselfness" [41]. Using an evolutionary algorithm, they evolved a representation of nonself space that did not assign a class, self or nonself, but rather assigned each point to a level of "selfness". This method provided flexibility in choosing a threshold of points to classify as self and points to classify as nonself.

Dasgupta and Gonzalez [24] extended this idea to fuzzy classifications. Dasgupta replaced the function that returned yes or no depending on whether a point was in a class or not, respectively. The new function returned a number in $[0, 1]$, which was a probability

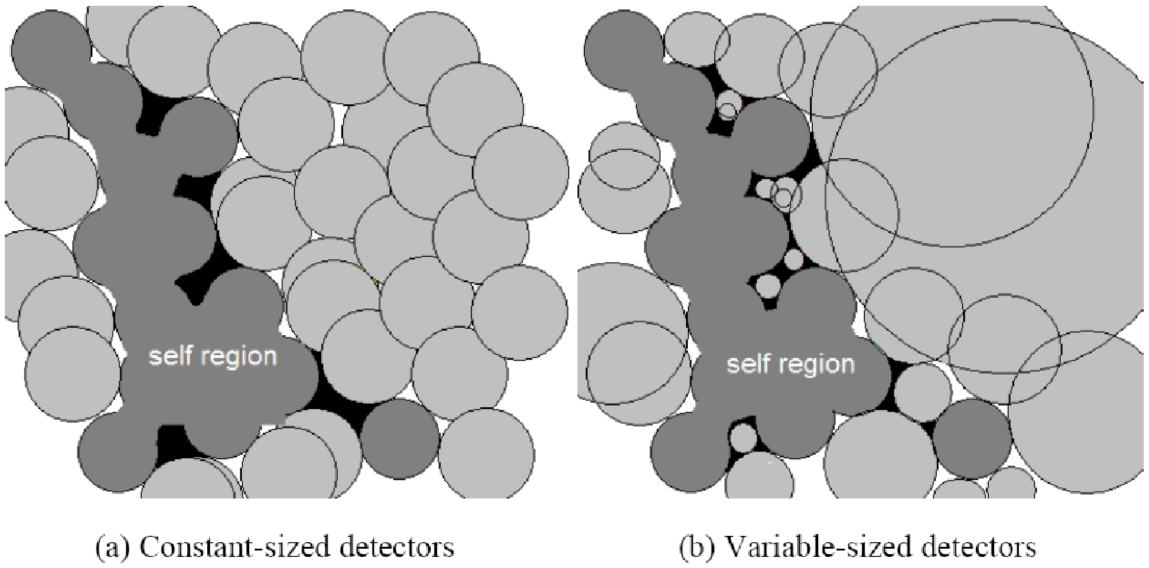


Figure 3: Covering nonself space with (a) constant-sized detectors and (b) variable-sized detectors. The black is uncovered space (as a result of the detectors not being granular enough), dark grey is self region and light grey is nonself detectors [40].

that a point was a member of self (or nonself). Using this technique, Dasgupta identified levels of membership in self. This was useful because different actions could be taken based upon the level of membership. For example, if a point had a high membership in self, it would most likely be ignored, although a note might be made in a log that its appearance occurred. On the other hand, a point with a very low membership in self might raise an alarm to email or contact via cell phone the system administrator.

Gomez et al. [38] improved on the fuzzy classification idea. They used fuzzy rules to describe the level of membership in self by a continuous probability, instead of by discrete levels. Experimental results showed improvement for this technique over the previous technique in which they assigned a level of membership to each point of the self-nonself space [24].

Other work has investigated ways of compacting the representation of self space. This has benefit potential because most published negative selection algorithms construct nonself detectors in the complementary space of self space. This is usually a bottleneck since self space is usually quite large and nonself detectors must be compared against every point in self space [56]. Hang et al. applied the generalized (matches more than one point)

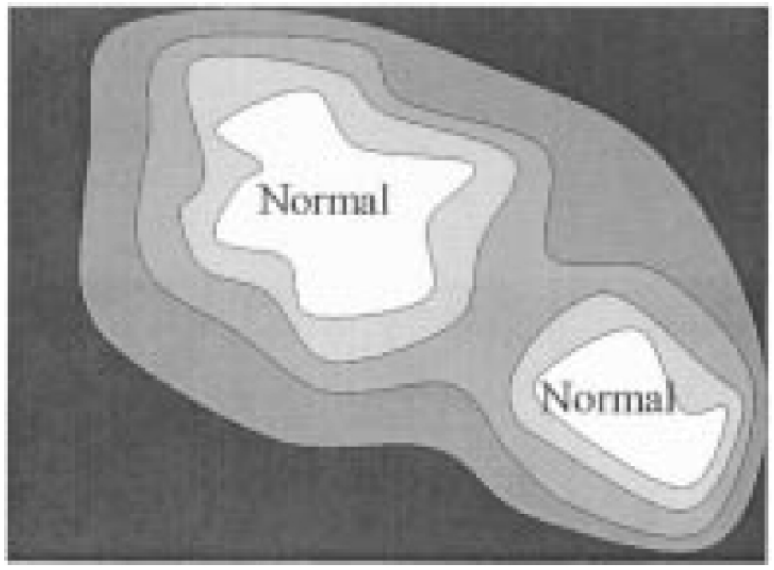


Figure 4: Different levels of membership in self [24].

nonself detector to self representations [43]. They suggested using self representations that represent multiple points in self space.

This seems obvious, but it had not been done before, probably because the BIS does not do this. However, the goal is not to build a BIS, but rather to apply good ideas from the BIS. Before running the negative selection algorithm to evolve good nonself detectors, they evolved a number of self representers by using a coevolutionary genetic algorithm (see the work of Potter and De Jong [76] for a short introduction to coevolution). They found that using generalized self space representations eliminated much of the cost of the traditional negative selection algorithm.

Negative selection is the most commonly researched AIS algorithm, with several research groups currently investigating its uses. Another AIS algorithm, clonal selection, is described in the next section.

2.6.1 Clonal Selection. De Castro and Von Zuben produced a clonal selection algorithm called CLONALG [19] that uses three parameters: (1) the number of members of the population that is to be cloned, (2) the number of clones to produce for one individual and (3) how much to mutate the new clones. Garrett improved on this work by producing a clonal selection algorithm that was parameter-free [36]. This was an improvement because no parameter tuning is necessary to find the best parameter settings. Garrett's clonal selection algorithm is adaptive, meaning that it tunes the parameters as it goes, adaptively setting parameters to optimal values. Garrett's adaptive algorithm produces results similar to CLONALG if it is allowed to have four times the number of fitness function evaluations that CLONALG gets in one run. Garrett believes that this is acceptable because four times the the number of evaluations is much less than the number required to tune the parameters in CLONALG over multiple runs.

Kim and Bentley [57] introduced a dynamic clonal selection with the following two important properties: (1) it learns normal behavior through exposure to only a small subset of self antigens at one time and (2) its detectors are replaced whenever previously observed normal behavior no longer represents current normal behaviors. The second property is especially important because it allows the artificial immune system to adapt. In the real world, an AIS in a computer must have the property of adaptability because the definition of normal behavior may change from day to day, as a result of many factors.

The AIS algorithms described in the preceding sections can be used to build classification systems, both the NID data and for other data. The following section presents a short discussion of research in the area of NID data classification issues.

2.7 Network Data Classification

The goal of a network data classification system for intrusion detection is to minimize type I and type II errors. Table 3 shows the relationship between true classification, assigned classifications, and error types. A type I error is a false alarm (i.e. an innocuous network datum classified as malicious). A type II error is a false negative (i.e. a malicious network datum classified as innocuous). However, these are conflicting goals. Lowering the type I error rate increases the type II error rate and vice versa. Although this is a common objective of all network intrusion detection systems, little work has been done to investigate and quantify this issue.

The most significant work in this area is from Khoshgoftaar et al. [53]. They extended the Expected Cost of Misclassification metric and introduced the Modified Expected Cost of Misclassification (MECM). They observed that the best NID system is not simply the one that minimizes the expected error. There are two reasons for this: (1) The costs of type I and type II errors are different. A type II error is more costly because that means that an intrusion has been missed and the harm caused by the intrusion could be severe. (2) A network administrator has limited resources (e.g. especially time) that can be spent responding to alarms. The MECM handles the first issue mentioned above by weighting type I and type II errors differently. The second issue is handled by penalizing classifications whose type I error rate requires more resources than are available to a system administrator for responding to the alarms.

Although the implications of network data classification are specific to NID data, analogous arguments can be applied to other data sets against which an AIS may be applied. The important point here is that the decision of which classifier must take into consideration problem domain information. The decision cannot be made simply on the basis of numerical results for maximization of true positive and minimization of false alarm rates.

Table 3: Table showing false positive and false negative errors. A false negative (type II) error occurs when a malicious network datum is classified as innocuous. A false positive (type II) error occurs when an innocuous network datum is classified as malicious.

		True Class	
		innocuous	**malicious**
Assigned Class	**innocuous**	correct	Type II (false negative)
	malicious	Type I (false positive)	correct

The NID problem is hard because intrusions can be so different. For this reason, some research efforts have investigated ways of using a human in combination with a machine learning technique, such as an AIS.

2.8 Human Interaction in Searching for Good Antibodies

Williams et al. [96] [97]used antibodies that matched more than one point in a binary self-nonself space. They compared several bit string matching metrics in an effort to identify one that worked well for his research [44]. They searched for good antibodies using a genetic algorithm. Their most innovative contribution, however, was the use of an interactive search.

Williams et al. introduced a system by which a human (system/network administrator or some party interested in the security of a network) could guide the search for good antibodies toward more likely areas of the search space. This is important because humans have knowledge that the genetic algorithm does not. Therefore, it is useful to combine the knowledge. Williams designed and implemented a GUI through which a human could choose three dimensions (features, etc.) of the search space and guide the search in a direction that seemed promising to the human. Although this work was, of course, not a final solution to the NID problem, it contributed a unique idea of how to combine the expertise of computers and humans in order to obtain a solution.

Any type of classification system that has been built for NID problems must be tested for validation that it works. The following section describes current NID testing practices.

2.9 Testing Network Intrusion Detection Systems

The field of network intrusion detection is currently receiving much attention. Many products have been produced in academia and it is desired to have some public metric against which products can be evaluated. This section provides background on available network intrusion test data sets and some techniques which have been used for network intrusion detection system testing.

2.9.1 Network Intrusion Data Sets. Although there are many parties actively researching the network intrusion detection problem, there are very few data sets available for testing network intrusion detection systems. There are two primary reasons for this: First, privacy-data cannot be simply pulled from a network for testing purposes, because network data contains personal information (emails, credit card number, etc.). Second, when data is pulled from a network, it is unlabeled. This makes it difficult to use for training and testing of intrusion detection systems because it is unknown what is an attack and what is not. One solution to these two problems is to simulate network data. However, simulation introduces the problem of how to assure simulated network traffic is statistically realistic. In addition to statistical errors, simulated network traffic is also likely to introduce other artifacts [63].

As a result of the reasons discussed in the previous paragraph, it is a difficult undertaking to produce a good simulation of network traffic data. The standard data sets for several years were produced by Lincoln Labs at MIT in projects funded by DARPA [61]. Although these data sets have enjoyed widespread usage for IDS training and testing, they suffer from the artifacts of simulation. Mahoney and Chan provide a good analysis of the Lincoln Labs data sets in [63].

2.9.2 Network Intrusion Detection System Testing Methods. Usually, a NID system is allowed to train on some clean data (no known attacks). Then, after training, the NID system is applied to a set of test code that has both clean and attack data. The effectiveness of the NID system is measured by the number of correct classifications (clean v. attack). A NID system is considered effective if it minimizes type I and type II errors (see Table 3).

Although this is the most common approach, other approaches exist. Some systems do not require training on clean data, so they are testing simply by presenting dirty data and evaluating classification success [31]. Dozier et al. [28] suggest a novel approach that uses an evolutionary algorithm to evolve "hackers." Their evolutionary algorithm gives higher fitness to attacks that cause more systems to break (misclassify the attack). In this way, their evolutionary algorithm identifies a small subset of attacks that is effective at discriminating between effective and ineffective NID systems.

Construction of an AIS to detect network intrusions (or some class in any data set) also requires a technique for building the detectors. Depending on the data set, this may or may not be a difficult problem. Often, the space of possible detectors is too large for exhaustive search and rugged enough that there is no known deterministic algorithm for building detectors. An evolutionary algorithm is a stochastic search technique that is often used in optimization problems.

2.10 Evolutionary Algorithms

Evolutionary algorithms are used to solve problems that have a large, random solution space. If the solution space is not very random, there is probably a deterministic approach to find an optimal solution, even though the solution space may be large. It is noted that there is not an Evolutionary algorithm that is best for all problems. The "best" evolutionary algorithm is problem dependent [98] [95].

Forrest et al. [35] investigated the usefulness of genetic algorithms as a tool for evolving good nonself recognizers in a computer immune system. One of their conclusions was that mutation in a GA works well for modelling the somatic mutation that occurs in the BIS when a new antibody is created and mutated through clonal selection. Crossover, on the other hand, is better for modelling the evolution of good antibodies that occur in a population over time. The previous sentence deserves a short explanation.

In the BIS, antibodies (B-cells and T-cells) are not randomly created. Rather, there is a library of good receptor building blocks (these building blocks have been effective in the past at recognizing antigens) from which receptors for new B-cells and T-cells are formed. These libraries of receptor building blocks have been evolving over many generations in the biological immune system. Forrest believes that this can be modelled in an AIS through the GA variation operator crossover.

Goldberg et al. [37] have introduced a genetic algorithm that explicitly manipulates building blocks. This algorithm is called the "messy GA" since it is a little messy to explicitly manipulate individual building blocks. Since the BIS uses building block libraries, the messy GA may be well suited for evolving good building block libraries in an AIS. This possibility is not addressed here by future research could investigate possible benefits from using the messy GA to evolve libraries for generating new antibodies.

2.11 Summary

This chapter defines the negative selection problem. Background is provided on the BIS roots of negative selection and on the NID problem, one of the most promising applications of negative selection. The negative selection algorithm is developed as a machine learning technique, including a sampling of different negative selection approaches. Following a short discussion of details specific to testing against NID data, a short discussion of evolutionary algorithms is provided. Evolutionary algorithms can be used to find solutions to intractable problems, such as generating detectors for network intrusion. The next chapter introduces a new type of detector for negative selection, the hyper-ellipse detector.

III. High Level Design

This chapter describes the design for an algorithm that generates hyper-ellipsoids (hereafter referred to as "ellipsoids") to cover nonself space in a negative selection problem. The first section justifies the selection of ellipsoids as a matching function for detectors. Section 3.2 presents a mathematical definition of ellipsoids. Next, Section 3.3 develops an algorithm to generate ellipsoid detectors. Section 3.4 provides an analysis of the time and memory complexity of the algorithms described in Section 3.3. Section 3.5 concludes this chapter and describes reasoning for lower level implementation details.

3.1 Ellipsoids as Detectors

Design of a negative selection algorithm requires the choice of a detector. A detector is defined by two things: (1) The function *match* (see Equation 1) and (2) the detector that is an argument to *match*. It is possible to build a *match* function to map to any set of points. However, a *match* function that returns structured sets of points is easier to manipulate than sets of points that are more random. Further, geometrically structured sets of points are intuitive because the output of *match* is always a connected space.

Also, it is easier to use a *match* function that is itself simple. A *match* function that returns high-entropy sets of points is often more complex than a *match* function that returns a hyper-sphere, for instance.

There are *match* functions that do not map to connected sets of space. These may be useful in a designing detectors for a negative selection problem. However, we choose to use a *match* function and detectors that return connected geometric spaces of points because previous research [49] [97] has shown that geometric structures perform well as detectors in a negative selection problem. Future research may investigate the feasibility of real-valued detectors that do not represent connected geometric structures.

We desire to improve on hyper-rectangle and hyper-sphere detectors by adding geometric flexibility. Flexibility means that the detector more accurately provides a fit to boundaries in self-nonself space. We surmise that a more flexible detector accomplishes similar (to a less flexible detector) classification accuracy while using fewer detectors. We choose ellipsoids because they are a generalization of hyper-spheres. Hence, ellipsoids retain the strengths of hyper-spheres while adding flexibility, intuitively conceptualized as "stretching" and "orienting." The next section provides a mathematical definition for ellipsoids. The mathematical definition is extended to understand the volume of an ellipsoid and membership of an arbitrary point in an ellipsoid.

3.2 Ellipsoids

This section provides a mathematical definition of ellipsoids, as well as solutions to two low level operations, volume and membership of an arbitrary inside of an ellipsoid.

3.2.1 Mathematical Definition of Ellipsoid. The mathematical definition of an ellipsoid is a straightforward generalization of the classical 2-d ellipse. Equation 13 defines the classical 2-d ellipsoid:

$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1, \tag{13}$$

where $(x_0,\ y_0)$ is the center of the ellipse and a and b are the lengths of the x and y semiaxes, respectively. Equation 14 shows a matrix formulation of Equation 13.

$$(\mathbf{x}-\omega)^T\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T(\mathbf{x}-\omega)=1 \tag{14}$$

Equation 14 defines the same 2-d ellipse as Equation 13 if $\mathbf{x}=\begin{bmatrix} x \\ y \end{bmatrix}$, $\mathbf{\Lambda}=\begin{bmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{bmatrix}$, $\omega=\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, and $\mathbf{V}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

This is shown in Equation **??**.

$$\left(\begin{bmatrix} x \\ y \end{bmatrix}-\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}\right)^T\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T\left(\begin{bmatrix} x \\ y \end{bmatrix}-\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}\right)=1 \tag{15}$$

ω is the center of the ellipsoid. The columns of matrix $\mathbf{V}$ are the orientations of the x and y semiaxes, respectively. In this case, $\mathbf{V}$ is the identity matrix because Equation

13 defines a 2-d ellipse whose semiaxes are oriented along the x and y coordinate axes. $\mathbf{\Lambda}$ defines the lengths of the ellipsoid semiaxes.

Equation 14 is also the generalized equation for an n-d ellipsoid if $\mathbf{x}$, ω are $n \times 1$ matrices and $\mathbf{V}$, $\mathbf{\Lambda}$ are $n \times n$ matrices. $\mathbf{\Lambda}$ is subject to the constraint that it must be diagonal with positive entries. $\mathbf{V}$ is subject to the constraint that its columns must be orthonormal. Figure 5 shows the $\omega, \mathbf{\Lambda}$ and $\mathbf{V}$ graphically for a 2d ellipsoid.

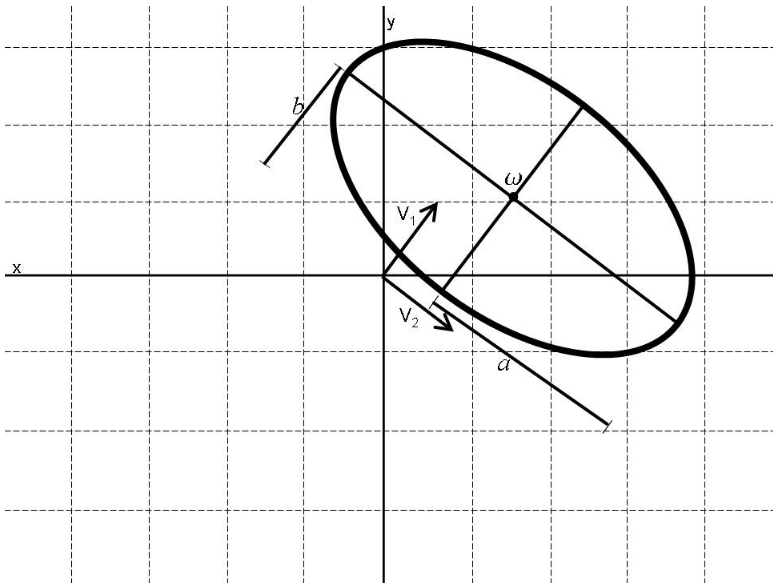


Figure 5: A graphical illustration of variables that define an ellipsoid.

There are three conceptual degrees of freedom for an n-d ellipsoid: center, semiaxis lengths, and orientations of semiaxes. ω is the center of the ellipsoid. The semiaxis lengths are the square roots of the inverses of the diagonals of $\mathbf{\Lambda}$. The length ℓ_i of the i^{th} semiaxis is defined by

$$\ell_i = \frac{1}{\sqrt{\Lambda_{i,i}}}. \tag{16}$$

The i^{th} semiaxis also has an orientation characteristic. The orientation of a semiaxis is a vector that is parallel to the actual semiaxis of the ellipsoid. The columns of $\mathbf{V}$ are the orientations of the semiaxes. Since $\mathbf{\Lambda}$ defines the lengths of the semiaxes, the columns of $\mathbf{V}$ are normalized so that they have length 1. Since the semiaxes of an ellipsoid are all orthogonal to each other, and the orientations are all length 1, $\mathbf{V}$ is also orthonormal. The i^{th} column of $\mathbf{V}$ is the orientation of the i^{th} semiaxis, whose length is $\ell_i = \frac{1}{\sqrt{\Lambda_{i,i}}}$.

It is easy to prove that the columns of $\mathbf{V}$ are the orientations of the semiaxes of an ellipsoid defined by Equation 14. Changing the orientation of the semiaxes means rotating the ellipsoid. A rotation in n-space is defined by a $n \times n$ orthonormal matrix. $\mathbf{V}$ is an orthonormal matrix, and defines this rotation. If $\mathbf{x} - \omega$ is a point on the surface of some ellipsoid e, then $\mathbf{V}(\mathbf{x} - \omega)$ is a point on the surface of an ellipsoid that has been rotated by $\mathbf{V}$. This is shown with the substitution $(\mathbf{x} - \omega) \Rightarrow \mathbf{V}(\mathbf{x} - \omega)$, leading to:

$$(\mathbf{V}(\mathbf{x} - \omega))^T \mathbf{V} \boldsymbol{\Lambda} \mathbf{V}^T (\mathbf{V}(\mathbf{x} - \omega)) = 1. \tag{17}$$

The definition of transposition in linear algebra states that $(\mathbf{V}(\mathbf{x} - \omega))^T \mathbf{V} = (\mathbf{V}^T \mathbf{V}(\mathbf{x} - \omega))^T$. Substituting this equality results in

$$(\mathbf{V}^T \mathbf{V}(\mathbf{x} - \omega))^T \boldsymbol{\Lambda} \mathbf{V}^T (\mathbf{V}(\mathbf{x} - \omega)) = 1 \tag{18}$$

Since $\mathbf{V}$ is orthonormal, $\mathbf{V}^T \mathbf{V} = \mathbf{I}$. Hence, Equation 18 simplifies to Equation 14, which is true because of the the assumption that $(\mathbf{x} - \omega)$ is on the surface of the ellipsoid. Rotation preserves the relative positions of points on the ellipsoid. Hence, if point $\mathbf{p}$ is on a semiaxis in the unrotated ellipsoid, then $\mathbf{Vp}$ is on the rotated ellipsoid. Therefore, the orientation of the semiaxes is the columns of $\mathbf{V}$.

3.2.2 Volume of an Ellipsoid. Since one of the design objectives of a negative selection algorithm is efficiency (see Chapter I), it is desirable that detectors cover as much nonself space as possible, making it useful to know the volume of an ellipsoid. Fortunately, the volume of an ellipsoid is obtained by evaluating a simple, exact expression. Tee [90] [88] shows that the volume of the n-d ellipsoid defined by Equation 14 is

$$V = \Omega_n \prod_{i=1}^{n} \frac{1}{\sqrt{\boldsymbol{\Lambda}_{i,i}}} \tag{19}$$

where $\mathbf{\Lambda}$ is retains its definition from Equation 14 and Ω_n is the volume of a n-d unit hyper-sphere. Smith and Vamanamurthy [81] show that the volume Ω_n of an n-dimensional unit hypersphere is

$$\Omega_n = \frac{\pi^{n/2}}{\Gamma(1 + \frac{1}{2}n)} \tag{20}$$

The Γ function is a mathematical extension of the factorial function from positive integers to real numbers. Java source code that computes the Γ function has been produced by several groups, including the PAL project [87]. Ω_n is computed for $n = 1...20$ once and then stored in a table. Table 4 shows the volume of the unit hyper-sphere through 20 dimensions.

Table 4: Volumes of unit hyper-sphere for up to 20 dimensions.

Dimensions	**Volume**
1	2.0000000000215654
2	3.1415926536524683
3	4.188790204831557
4	4.934802200643129
5	5.263789013971081
6	5.167712780153064
7	4.724765970382346
8	4.058712126497739
9	3.298508902774271
10	2.5501640399282204
11	1.8841038794102152
12	1.3352627688812284
13	0.9106287547931023
14	0.5992645293244241
15	0.3814432808246492
16	0.23533063035939014
17	0.14098110691732263
18	0.08214588661119586
19	0.046621601030113424
20	0.025806891390023085

The interpretation of Equation 19 is that the volume of a n-d ellipsoid is the volume of a n-d unit hyper-sphere multiplied by the semiaxis lengths (see Equation 16) of the ellipsoid. This is intuitive, indicating that "stretching" a unit hyper-sphere by changing the length of one the semiaxes is analagous to stretching the volume my multiplying by the same scalar that is used to "stretch" the semiaxis.

3.2.3 Is a Point Inside of an Ellipsoid? A negative selection algorithm needs to be able to determine whether or not a detector matches a point. If the detector is an ellipsoid, a detector matches a point if the point is inside of the ellipsoid represented by the detector. Kelly et al. [51] report that the squared Mahalanobis distance can be used to determine whether or not a n-d point $\mathbf{p}$ lies inside of ellipsoid e. Kelly et al. choose a value ρ and if the squared Mahalanobis distance (left side of Equation 21) is less than ρ, then the point lies within the ellipsoid. They are effectively scaling the ellipsoid when they choose the value ρ. We choose $\rho = 1$. $\mathbf{p}$ is inside of e if and only if the inequality in Equation 21 holds.

$$(\mathbf{p} - \omega)^T \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T (\mathbf{p} - \omega) < 1 \tag{21}$$

As this operation is performed often in our algorithm, we present the complexity of computing the Mahalanobis distance.

3.2.3.1 Complexity of Squared Mahalanobis Distance. Squared Mahalanobis distance requires evaluation of the expression, $(\mathbf{x} - \omega)^T \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T (\mathbf{x} - \omega)$. However, a simplification is possible. If the Mahalanobis distance is to be computed for multiple points against one ellipsoid, we can compute $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ only once and then use Equation 22 for the Mahalanobis distance.

$$(\mathbf{p} - \omega)^T \mathbf{A} (\mathbf{p} - \omega) < 1 \tag{22}$$

Evaluation of the left side of Equation 22 requires two vector subtractions, a matrix transpose, and two matrix multiplies. x is an $n \times 1$ matrix and A is a $n \times n$ matrix. Hence, the complexity of computing the Mahalanobis distance is determined by

- two vector subtractions: $2n$ operations
- transpose of $[n \times n]$ matrix: n^2 operations
- matrix multiply of a $[1 \times n]$ by a $[n \times n]$ matrix: n^2 operations

- matrix multiply of a $[1 \times n]$ by a $[n \times 1]$ matrix: n operations

Hence, the bounded complexity of computing the Mahalanobis distance is given by Equation 23

$$O(2n + n + n^2 + n) = O(n^2) \tag{23}$$

3.3 An Evolutionary Algorithm to Evolve a Set of Ellipsoids

With a solid ellipsoid model established, the next step is to produce a set of ellipsoids optimized according to Equations 2 and 3. Producing a set of ellipsoids that satisfies these constraints is not a trivial problem. There is no known deterministic algorithm to accomplish this goal, completing in a tractable amount of time. Also, performing an exhaustive search for an optimal solution is not an option because the problem space is prohibitively large. If each coordinate axis is discretized into α points, exhaustive search requires searching a space size $O(\alpha^{2n+1}n^2)$, where α^n is the number of possible center points, $O(\alpha n)$ is a bound on the number of combinations of axis lengths, and $O(n\alpha^n)$ is a bound on the number of possible orientations. Further, the previous complexity analysis makes the unreasonable assumption that there is only one ellipsoid. As α is a reasonably sized number for the discretization of coordinate axes, exhaustive search is obviously infeasible.

An exhaustive search is not be feasible unless computational technology progresses many orders of magnitude beyond current limitations. However, there may be some merit in the development of deterministic algorithms for ellipsoid generation. Deterministic algorithms would be an excellent direction for future research.

In this research we use an evolutionary algorithm because the problem of generating ellipsoid detectors appears to be nonlinear and have stochastic components [17]. Evolutionary algorithms (EAs) have proven themselves in the optimization of hard problems with rugged, random solution spaces. The algorithm described in this section also uses some components of an algorithm called simulated annealing.

One of the first decisions that must be made in order to implement an EA is the representation of an individual and a population. Since a classic EA finds an optimal individual, the intuitive decision is to let an individual be a set of ellipsoids. Then, the individual (set of ellipsoids) that covers the most nonself space is chosen as the optimal solution. However, ellipsoid computations are costly, so having an entire population of sets of ellipsoids requires is a computational bottleneck. For this reason, we let an individual be one ellipsoid. The solution is then all (or a subset) of the individuals in the population. We describe an EA that reflects this design decision in variation operators (crossover and mutation), objective function, and convergence technique/criteria.

Evolutionary algorithms generally employ crossover and mutation variation operators to search the solution space. Objective functions evaluate the goodness of solutions. Convergence techniques can lead the algorithm to an "optimal" solution. This section addresses the mapping of the ellipsoid model into crossover, mutation, objective function, and a convergence technique in the evolutionary algorithm domain.

3.3.1 Crossover With Ellipsoids. The virtue of crossover in an EA is based on the fact (or belief) that individuals contain different "building blocks" that can be combined to produce a better individual than the "parents" that are crossed over. However, if "good" building blocks do not exist, then crossover is void of virtue.

Crossover works well when an EA evolves a population of individuals with the objective of choosing the best individual from the population when the algorithm completes execution. In this problem of ellipsoid detector generation, an individual would be a set of set of ellipsoids. After the EA has run to completion, the individual (set of ellipsoids) with the best objective value is chosen.

With this EA implementation, crossover could be performed by "trading" of ellipsoids between different individuals. However, we do not use this pure EA method because of the computational burden associated with maintaining a population that consists of multiple individuals, each of which consists of multiple ellipsoids.

The approach that we take is to maintain only one set of ellipsoids. The final solution is this set of ellipsoids when the algorithm has run to completion. Since only one complete individual is maintained by the EA, crossover does not make sense. Crossover in this case would be similar to choosing two portions of the representation of the same individual and switching them.

Figures 6 and 7 illustrate the reason we choose not to use crossover. Figure 6 shows that crossover is useful if a population is a set of individuals, each of which is a set of ellipsoids. In this situation, building blocks are individual ellipsoids. Figure 6 shows that crossover is simply a trade of ellipsoids. This trade makes sense according to the definition of crossover and Figure 6 shows how this can lead to a better child, which is the objective of crossover in an EA.

Figure 7 shows the results of crossover when an individual is one ellipsoid. The intuitive way to accomplish crossover in this case (there may be others) is to let a building block be ω, $\mathbf{\Lambda}$, or $\mathbf{V}$, which is the center point, semiaxis lengths, or semiaxis orientations, respectively. However, since all of the ellipsoids are actually part of one solution, these are not really "homologous" fragments of the chromosome. Some of the resulting "bad" children are shown in Figure 7. These figures and arguments do not prove, of course, that crossover is not useful for our one-solution-in-population approach. The preceding arguments do, however, provide an intuitive explanation of why the algorithm developed in this research does not employ crossover.

3.3.2 Mutating an Ellipsoid. EAs can employ mutation as a variation operator that searches the solution space. There are two design goals for mutation. First, it should be possible to mutate any valid ellipsoid to any other valid ellipsoid through a finite series of mutations. This design goal is especially important because we are not using crossover. This means that mutation is the only variation operator (other than random generation of new ellipsoids). Second, the mutation should be random but should also not be too far from the unmutated ellipse. If a mutation is too far, then mutation works like a random search, instead of an opportunity to improve on good ellipsoids. "Too far" is, of course, a subjective term. For lack of a better definition, the following is used: The mutated ellipsoid overlaps "most" of the unmutated ellipsoid.

Mutating an ellipsoid is not as simple as one might think ("just stretch it, spin it, bump it!"). One method might be to simply randomly change the ω, $\mathbf{V}$, and $\mathbf{\Lambda}$ from

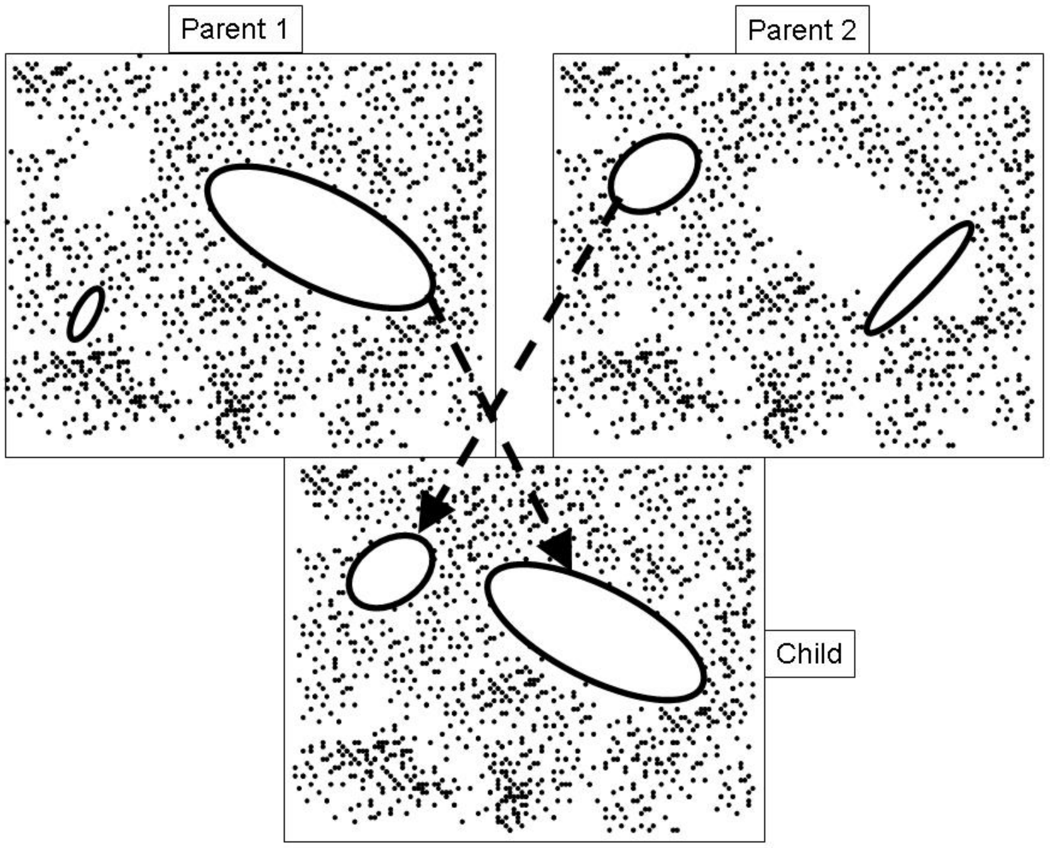
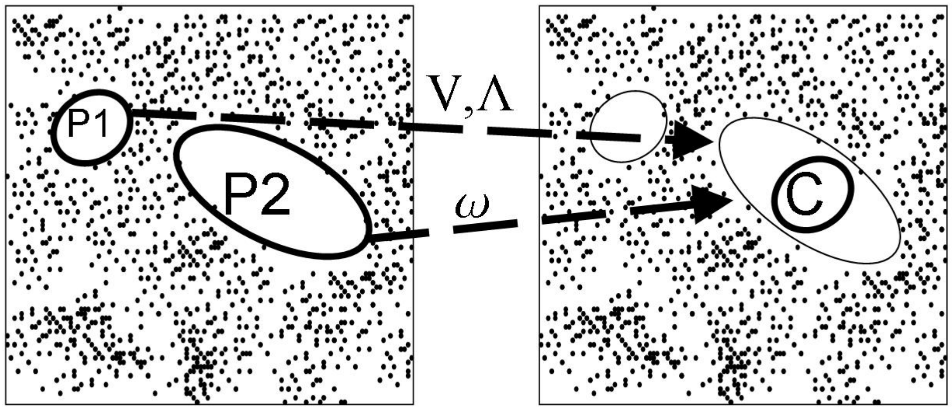
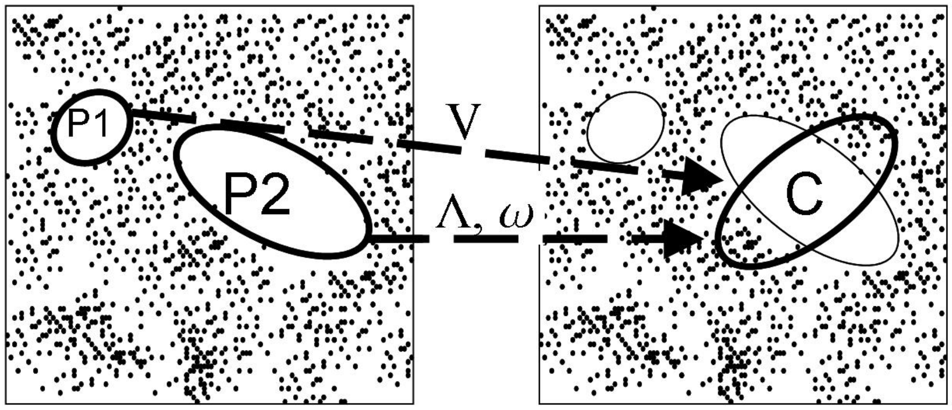


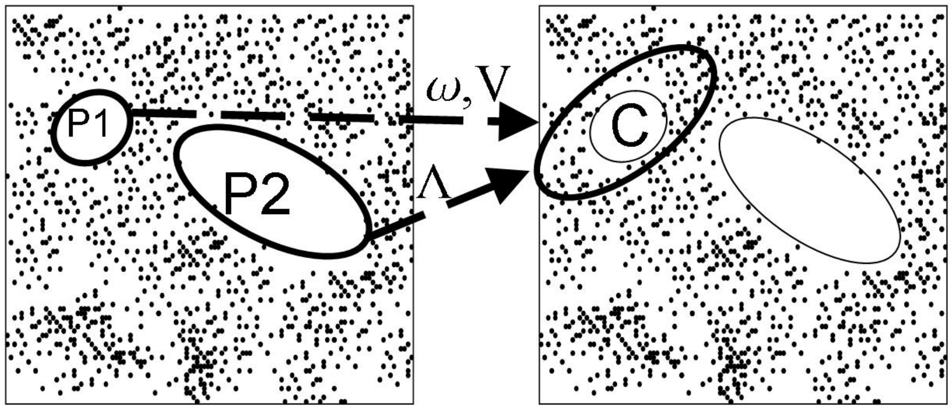
Figure 6: The two individuals on top are the parents. Each parent is a set of two ellipsoids. The obvious optimal solution to this problem is two ellipsoids. The arrows show how two "building blocks" (a "building block" is an ellipsoid) can be combined through crossover to produce a good child, which is the optimal solution.



(a) Center



(b) Orientation



(c) Semiaxis Lengths

Figure 7: Crossover problems when an individual in the population is an ellipsoid and not a set of ellipsoids. ω, $\mathbf{\Lambda}$, and $\mathbf{V}$ retain their definitions from Equation 14. $P1$ and $P2$ denote "parent 1" and "parent 2", respectively. "C" labels the bold ellipsoid it is inside of, and is the child. The labels ω, $\mathbf{\Lambda}$, and $\mathbf{V}$ on the arrows denote the "building blocks" coming from each parent.

Equation 14. A random change to ω results in a valid ellipsoid, but $\mathbf{V}$ and $\mathbf{\Lambda}$ are subject to constraints. $\mathbf{V}$ must have orthonormal columns and $\mathbf{\Lambda}$ must be diagonal with positive values. Random changes to $\mathbf{V}$ and $\mathbf{\Lambda}$ have a high probability of violating their respective constraints. For this reason, we choose to employ a different technique for ellipsoid mutation.

Although it does not make sense to accomplish ellipsoid mutation by randomly changing ω, $\mathbf{V}$ and $\mathbf{\Lambda}$, these three parameters that define an ellipsoid do provide insight into a good way to mutate an ellipsoid. ω, $\mathbf{V}$ and $\mathbf{\Lambda}$ represent three independent characteristics of an ellipsoid: center, semiaxis lengths, and semiaxis orientations. The following sections describe methods for performing valid mutations of ω, $\mathbf{V}$ and $\mathbf{\Lambda}$. The mutations described in the following sections also conform to the two design goals: (1) possible to reach any ellipsoid beginning from any ellipsoid through a series of mutations (2) mutation is small.

Center Mutation. The center of an ellipsoid defined by Equation 14 is ω. Let $M_c : \{\omega | \omega \text{ is a } n \times 1 \text{ vector}\} \Rightarrow \{\omega | \omega \text{ is a } n \times 1 \text{ vector}\}$ be the EA center mutation operator. Any $n \times 1$ vector returned by M_c is valid. However, fulfillment of the second design goal requires that ω^m, the mutated center, remain near ω. In order to maintain the ellipsoids in useful positions, we add the constraint that ω^m cannot be outside of the problem domain bounds. Allowing the center to be outside of problem domain bounds could result in an ellipsoid that can never cover nonself space because it is impossible for a nonself point to exist where the ellipsoid is.

Two schemes are investigated for center mutation. The simplest mutation technique randomly chooses one dimension and mutates the corresponding component of the center point. If the i^{th} dimension is randomly chosen, M_c chooses ω_i^m from a Gaussian distribution with mean ω_i. The standard deviation for the distribution is a parameter that can be changed. This method is intuitive, simple to implement, and fulfills the two mutation design goals specified above. Any center point can be reached from any starting center point through a series of mutations. The mutation is only a small change because the Gaussian distribution keeps most points near the mean. We keep ω^m within the problem domain bounds by setting ω^m to be on the boundary if the value chosen from the Gaussian distribution is outside of the problem domain bounds.

Although it seems that this center mutation technique should work well, it suffers from a drawback. There is a situation in which a center mutation in a direction other than along a semiaxis is desirable, but mutation along any of the semiaxes is undesirable because it causes the ellipsoid to cover one or more self points. Such a situation is shown is Figure 8. This weakness causes an ellipsoid to "get stuck" before it has evolved into an optimal ellipsoid. Because of this, we choose to not use this center mutation technique.

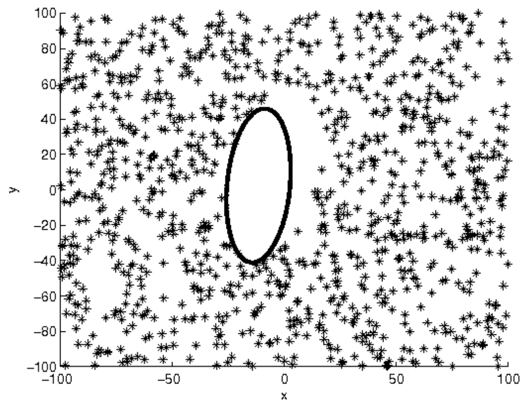


Figure 8: An ellipsoid surrounded by self points. The ellipsoid shown cannot be moved along only one semiaxis without causing the ellipsoid to cover self point(s).

However, the described weakness does not render the mutation technique useless. A simple extension is to repeat the mutation for all dimensions. This allows the center to be mutated in any direction, solving the weakness shown in Figure 8.

The described center mutation fulfills the design goals because there is a nonzero probability of returning any point when mutating ω. Also, the Gaussian distribution guarantees that ω^m is usually near ω.

Orientation Mutation. The orientation of an ellipsoid defined by Equation 14 is $\mathbf{V}$, whose orthonormal columns are the orientations of each individual semiaxes. Let $M_o : \{v|v \text{ is a } n \times n \text{ matrix}\} \Rightarrow \{v|v \text{ is a } n \times n \text{ matrix}\}$ be the EA orientation mutation operator. Any $n \times n$ orthonormal matrix returned by M_o is valid. However, fulfillment of the second design goal requires that $\mathbf{V}^m$, the mutated semiaxis orientations, must be restrained by some bounds.

Changing the orientation of an ellipsoid is simply rotation of the ellipsoid. In two dimensions, a mutation for the orientation is easy to conceptualize. Figure 9 shows such a mutation.

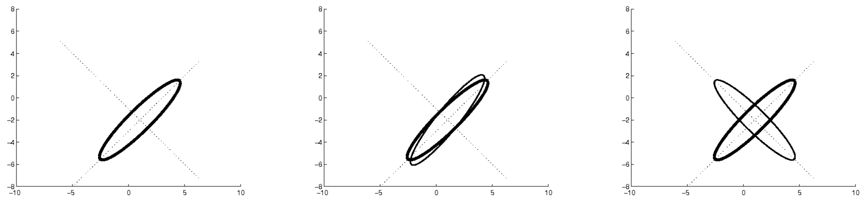


Figure 9: (a) shows an unmutated ellipsoid. (b) shows semiaxis mutation in the thin line. (c) shows an unacceptable (except in rare instances) mutation in the thinner line. In all figures, the semiaxes of the unmutated ellipse are shown in the heavier line.

The EA accomplishes orientation mutation by rotating the ellipsoid in a 2d plane. Let $\mathbf{v}_1$ and $\mathbf{v}_2$ be the vectors of two semiaxes that are chosen at random from the n semiaxes of the ellipsoid. That means that $\mathbf{v}_1$ and $\mathbf{v}_2$ are two of the columns from the matrix $\mathbf{V}$ (see Equation 14). In the following discussion, it is important to note that $\mathbf{V}$ is orthonormal so $||\mathbf{v}_1|| = ||\mathbf{v}_2|| = 1$. A plane η is defined by all points that are a linear combination of $\mathbf{v}_1$ and $\mathbf{v}_2$. $\mathbf{v}_1$ and $\mathbf{v}_2$ are orthogonal to each other because they are semiaxes of an ellipsoid. $\mathbf{v}_1$ and $\mathbf{v}_2$ are mutated so that they stay in plane η and are still orthogonal to each other after the mutation. Let $\mathbf{v}_1^m$ and $\mathbf{v}_2^m$ be $\mathbf{v}_1$ and $\mathbf{v}_2$, respectively, after having undergone mutation. $\mathbf{v}_1^m$ and $\mathbf{v}_2^m$ are orthogonal to the $n-2$ unchosen semiaxes because the $n-2$ unchosen semiaxes are orthogonal to η and $\mathbf{v}_1^m$ and $\mathbf{v}_2^m$ are in η after the mutation.

This is also shown mathematically. Let $\mathbf{v}_1^m = \alpha_1\mathbf{v}_1 + \beta_1\mathbf{v}_2$ and let $\mathbf{v}_2^m = \alpha_2\mathbf{v}_1 + \beta_2\mathbf{v}_2$. Let $\mathbf{u}$ be any one of the $n-2$ unchosen semiaxes. Two vectors are orthogonal if and only if their product $= 0$. $\mathbf{u}^T\mathbf{v}_1 = 0$ and $\mathbf{v}^T\mathbf{v}_2 = 0$ because $\mathbf{u}$ is orthogonal to v_1 and v_2. The following expressions show that $\mathbf{u}$ is orthogonal to $\mathbf{v}_1^m$ and $\mathbf{v}_2^m$ because the dot products $\mathbf{u}^T\mathbf{v}_1^m$ and $\mathbf{u}^T\mathbf{v}_2^m$ are equal to 0.

$$\mathbf{u}^T\mathbf{v}_1^m = \mathbf{u}^T(\alpha_1\mathbf{v}_1 + \beta_1\mathbf{v}_2) \tag{24}$$

$$= \mathbf{u}^T \alpha_1 \mathbf{v}_1 + \mathbf{u}^T \beta_1 \mathbf{v}_2 \tag{25}$$

$$= \alpha_1 \mathbf{u}^T \mathbf{v}_1 + \beta_1 \mathbf{u}^T \mathbf{v}_2 \tag{26}$$

$$= \alpha_1 0 + \beta_1 0 = 0 \tag{27}$$

so $\mathbf{u}$ and $\mathbf{v}_1^m$ are orthogonal. Substituting $\mathbf{v}_2$ and $\mathbf{v}_2^m$ for $\mathbf{v}_1$ and $\mathbf{v}_1^m$, respectively, shows that $\mathbf{u}$ is orthogonal to $\mathbf{v}_2^m$.

To accomplish this rotation, a small angle θ is chosen. The vectors that represent the randomly chosen semiaxes, s_1 and s_2, are both rotated by θ to produce new semiaxes, whose vectors are s_1^m and s_2^m. Since one of the design goals of orientation mutation is that the mutation be small, or local, a Gaussian distribution is chosen with the mean 0 and standard deviation $\frac{\pi}{2}$ radians. Figure 10 shows the PDF for choosing different angles. In practice, a parameter that increases or decreases the standard deviation (widens or narrows the curve in Figure 10) is added that allows the Gaussian distribution to be widened or narrowed.

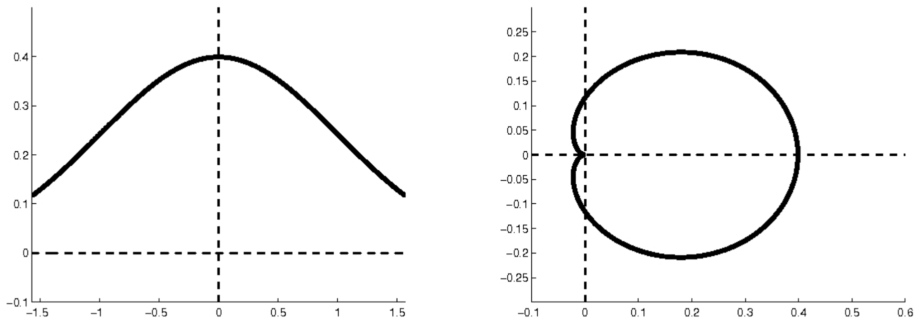


Figure 10: In (a), the x axis is the degree (in radians) of rotation and the y axis is the probability of randomly choosing that degree in a mutation. (b) shows the same thing in a polar representation. Each point on the curve represents the probability of randomly choosing a rotation of the angle between the x axis and the vector that represents the point.

v_1^m and v_2^m are computed as follows:

$$v_1^m = \cos(\theta) v_1 + \sin(\theta) v_2 \tag{28}$$

$$v_2^m = -\sin(\theta)v_1 + \cos(\theta)v_2 \tag{29}$$

If the dot product of v_1^m and v_2^m is 0, (i.e.$((v_1^m)^T v_2^m) = 0$, then v_1^m and v_2^m are orthogonal.

$$(v_1^m)^T v_2^m = [(\cos(\theta)v_{1_1} + \sin(\theta)v_{2_1}) \cdots (\cos(\theta)v_{1_1} + \sin(\theta)v_{2_1})] \begin{bmatrix} (-\sin(\theta)v_{1_1} + \cos(\theta)v_{2_1}) \\ \vdots \\ (-\sin(\theta)v_{1_1} + \cos(\theta)v_{2_1}) \end{bmatrix} \tag{30}$$

$$= \sum_{i=1}^{n} (\cos(\theta)v_{1_i} + \sin(\theta)v_{2_i}) * (-\sin(\theta)v_{1_i} + \cos(\theta)v_{2_i}) \tag{31}$$

$$= \sum_{i=1}^{n} -\cos(\theta)\sin(\theta)v_{1_i}^2 + \cos^2(\theta)v_{1_i}v_{2_i} - \sin^2(\theta)v_{1_i}v_{2_i} + \cos(\theta)\sin(\theta)v_{2_i}^2 \tag{32}$$

$$= \sum_{i=1}^{n} \cos(\theta)\sin(\theta)(v_{2_i}^2 - v_{1_i}^2) + (\cos^2(\theta) - \sin^2(\theta))v_{1_i}v_{2_i} \tag{33}$$

$$= \sum_{i=1}^{n} \cos(\theta)\sin(\theta)(v_{2_i}^2 - v_{1_i}^2) + \sum_{i=1}^{n} (\cos^2(\theta) - \sin^2(\theta))v_{1_i}v_{2_i} \tag{34}$$

$$= \cos(\theta)\sin(\theta) \sum_{i=1}^{n} (v_{2_i}^2 - v_{1_i}^2) + (\cos^2(\theta) - \sin^2(\theta)) \sum_{i=1}^{n} v_{1_i}v_{2_i} \tag{35}$$

Since s_1 and s_2 are orthogonal,

$$v_1^T v_2 = \sum_{i=1}^{n} v_{1_i}v_{2_i} = 0 \tag{36}$$

Also, if v_1 and v_2 are the same length (i.e. $||v_1|| = ||v_2||$)

$$\sum_{i=1}^{n} v_{1_i}^2 = \sum_{i=1}^{n} v_{2_i}^2 \tag{37}$$

so that

$$(v_{2_i}^2 - v_{1_i}^2) = 0 \tag{38}$$

Substituting Equations 36 and 38 in Equation 34,

$$\cos(\theta)\sin(\theta)\sum_{i=1}^{n} 0 + (\cos^2(\theta) - \sin^2(\theta))\sum_{i=1}^{n} 0 = 0 \qquad (39)$$

so that $(v_1^m)^T v_2^m = 0$. Hence, v_1^m and v_2^m are orthogonal.

The described orientation mutation meets the two design goals. It is possible to reach any orientation through a series of mutations in which only two semaixes are mutated at once (see Theorem 1. Also, the mutation is random and "small." This is accomplished by choosing the rotation angle from a Gaussian distribution. The standard deviation of the Gaussian distribution controls the amount of rotation.

Theorem 1. *It is possible to reach any orientation starting from any orientation through a series of mutations that mutate only two semiaxes at once.*

Proof. Define F_i to be an orthonormal $n \times n$ matrix. Let F_f be the final or target orthonormal $n \times n$ matrix. There exists L, a $n \times n$ matrix whose columns each represent the linear combination of columns of F_i necessary to obtain the corresponding column of F_f. L exists because F_i spans n-space, so the vector in each column in F_f is representable by a linear combination of columns of F_i. L is orthonormal because $L = F_i^{-1}F_f$ so $LL^T = F_i^{-1}F_f(F_i^{-1}F_f)^T = F_i^{-1}F_fF_f^T(F_i^{-1})^T = F_i^{-1}I(F_i^{-1})^T = I$.

The mutation operator works like a right matrix multiplication. If j and k are the indices of the chosen semiaxes, and letting the randomly chosen angle have a cosine of α and a sine of β, the mutation operator is equivalent to multiplying by $\mathbf{Y} = \mathbf{I}$ with the exception that $\mathbf{Y}_{jj} = \alpha$, $\mathbf{Y}_{kj} = \beta$, $\mathbf{Y}_{jk} = -\beta$, $\mathbf{Y}_{kk} = \alpha$. A matrix in the form of Y can perform any row operation. Since L is orthonormal, there is a series of row operations such that the result is I. If this series of row reductions is $Y_1, Y_2, ..., Y_z$, where z is the number of row operations necessary to get I, then $\prod_{i=1}^{z}(Y_i)^-1 = L$. Since L exists for any orthonormal matrix F_f, there exist mutations $Y_1^{-1}, Y_2^{-1}, ..., Y_z^{-1}$ that can produce F_f. □

Semiaxis Length Mutation. The length of the i^{th} semiaxis of the ellipsoid defined by Equation 14 is $1/\sqrt{\Lambda_{i,i}}$ (see Equation 16). Let $M_\ell : \{\lambda | \lambda \text{ is a } n{\times}n \text{ positive diagonal matrix}\} \Rightarrow \{\lambda | \lambda \text{ is a } n{\times}n \text{ positive diagonal matrix }\}$

be the EA semiaxis length mutation operator. Any $n \times n$ positive diagonal matrix returned by M_ℓ is valid. Let $\mathbf{\Lambda}^m$ be the mutated $\mathbf{\Lambda}$. Fulfillment of the second design goal requires that $\mathbf{\Lambda}^m_{i,i}$ be "close" to $\mathbf{\Lambda}_{i,i}$ for all $1 \leq i \leq n$.

M_ℓ mutates each of the n semiaxis lengths individually. Remember that ℓ_i is the length of the i^{th} semiaxis (see Equation 16. Let ℓ^m_i be ℓ_i after mutation. The semiaxis length mutation operator chooses ℓ^m_i from a Gaussian distribution with mean ℓ_i. The standard deviation is a parameter value that can be set to reflect the desired variability of the mutation. The algorithm also performs a check to ensure that $\ell^m_i > 0$.

This semiaxis length mutation fulfills the design goals because there is a nonzero probability of returning any valid $\mathbf{\Lambda}$ and the Gaussian distribution guarantees that the mutated semiaxis lengths are usually close to the unmuated lengths.

The described mutations for ω, $\mathbf{\Lambda}$, and $\mathbf{V}$ fulfill the design requirements of reachability and proximity. The mutation operator employed by the EA uses all three of these mutation operators to accomplish mutation. Reachability not only holds for ω, $\mathbf{\Lambda}$, and $\mathbf{V}$, but also for the entire ellipsoid. This is true because an ellipsoid is defined by the ω, $\mathbf{\Lambda}$, and $\mathbf{V}$ together. Since mutation allows each of ω, $\mathbf{\Lambda}$, and $\mathbf{V}$ to reach any valid value independent of the other two, an ellipsoid can reach any valid combination of values of ω, $\mathbf{\Lambda}$, and $\mathbf{V}$. Hence, defining ellipsoid mutation as a combination of center, semiaxis orientation, and semiaxis length orientation preserves reachability.

The other ellipsoid mutation design objective, proximity, is not so easily proved. Although it is possible to provide an analysis of the probabilities for overlap amount with the unmutated ellipsoid as a function of the mean and standard deviations of the Gaussian probability distributions used in the ω, $\mathbf{\Lambda}$, and $\mathbf{V}$ mutations, it is outside the scope of this research. We submit only a proof by intuition: Because the three individual mutations keep the mutated ellipsoid "near", the conglomerate mutation also maintains the closeness of the mutated ellipsoid.

3.3.3 Objective Function. An EA requires, in the least, a variation operator and a function to measure the goodness of individuals. With a good mutation operator defined, the next step is to develop the objective function, which measure the quality of individuals in a population. The output of the objective function affects whether or not the EA perpetuates an individual to the next generation. For this reason, the objective function should reflect as accurately as possible the true objective of the problem at hand.

The true objective of this problem is to cover as much space as possible while covering as few self points as possible. This objective divides naturally into a reward function and a penalty function. The reward function rewards ellipsoids for covering space. The penalty function penalizes ellipsoids for covering self points. The objective function is the difference of the reward function and the penalty function.

Reward Function. The reward function should encourage maximum coverage of space with a minimum number of ellipsoids. Since the solution to this problem is a set of ellipsoids, the reward value of each individual ellipsoid should reflect its contribution to the performance of the set of ellipsoids. The cumulative reward value of the population should be proportional to the amount of nonself space covered. If it is not, then the reward function does not accurately reflect the fulfillment of the objective.

Let E be a population of ellipsoids. Let

$$REWARD : \{\pi | \pi \text{ is a set of ellipsoids }\} \times E \rightarrow [0.00, 1.00] \tag{40}$$

be the reward function. i.e., the reward function maps a set of ellipsoids (the population) and an individual from the population to a value in $[0.00, 1.00]$. $\sum_{e \in E} REWARD(E, e)$ should be proportional to the area covered by the E. This means that when two or more ellipsoids in E overlap, only one of the ellipsoids should receive a reward for covering that area. This presents a difficult problem: Computing the total area covered by a set of ellipsoids. Also, in order to make decisions about which ellipsoids should perpetuate to future generations, the total area covered by a set of ellipsoids must be divided among the ellipsoids. Several options are investigated for the reward function.

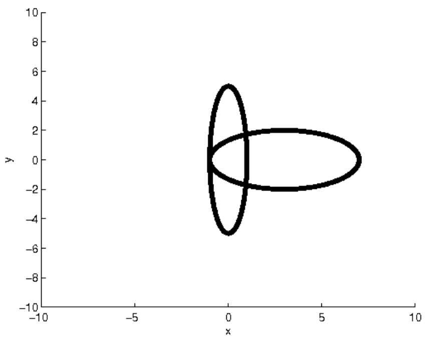
The first reward technique is based on an assumption about the behavior of evolving ellipsoids. The assumption is: If two ellipsoids overlap they one of them should be penalized because it is covering space that is already being covered. It is not helping to accomplish the goal of the set of ellipsoids, which is to collectively cover as much space as possible. In this approach to the $REWARD$ function, the $REWARD$ function is divided into two parts: $REWARD_r$ and $REWARD_p$. $REWARD_r$ rewards an ellipsoid for being large, i.e. covering up space. $REWARD_p$ penalizes an ellipsoid if it overlaps an ellipsoid that is larger than itself. So, $REWARD = REWARD_r - REWARD_p$. Equation 19 shows how to compute the volume of an ellipsoid. However, computing overlap between two ellipsoids is a trickier task. We investigate several approaches for approximating ellipsoid overlap.

For the general case, it is nontrivial to decide whether or not two ellipsoids intersect. If e_1 and e_2 are n-d ellipsoids, then figuring out where they intersect requires solving the following system of matrix equations:

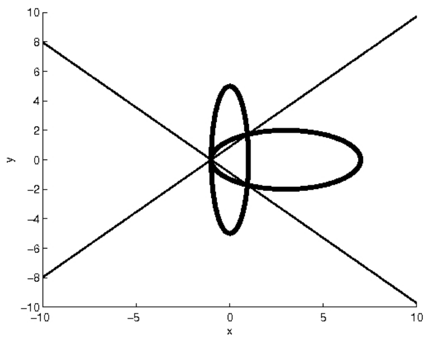
$$\begin{aligned} (\mathbf{x} - \omega_{\mathbf{1}})^T \mathbf{V_1} \mathbf{\Lambda_1} \mathbf{V_1}^T (\mathbf{x} - \omega_{\mathbf{1}}) = 1 \\ (\mathbf{x} - \omega_{\mathbf{2}})^T \mathbf{V_2} \mathbf{\Lambda_2} \mathbf{V_2}^T (\mathbf{x} - \omega_{\mathbf{2}}) = 1 \end{aligned} \tag{41}$$

41 is a nonlinear system of equations of matrices. If one ignores the fact that it uses matrices and vectors, it is a nonlinear system. When the complexity of multiple dimensions in the matrices is included, the complexity is increased. There is not a lot of literature on this subject. It suffices to say that analytically finding the intersection and the volume of the intersection between p and q is far from trivial. Tee [89] believes that finding the volume of the intersection is difficult: "Even in 2 dimensions that is a nontrivial problem, and in 3 dimensions, it is quite difficult." For this reason, we choose to use an approximate method to estimate the amount of overlap between two ellipsoids.

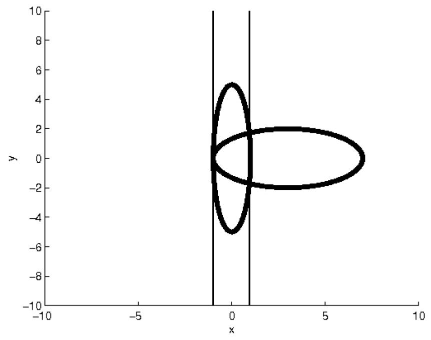
2d Ellipsoids. It is possible to obtain an analytical solution for the intersection of two ellipsoids in two dimensions. Hill [73] shows how to find the intersection points of two 2d ellipsoids using degenerate conics. Hill shows that the intersection of the surfaces of two ellipsoids is always a conic. Figure 11 shows the degenerate situation that occurs when ellipsoids intersect at exactly three points. Although this method is simple and it works well, it does not scale to higher dimensions. If the points of intersection are known in two dimensions, piecewise integration can be used to analytically find the volume of overlap.



(a) A



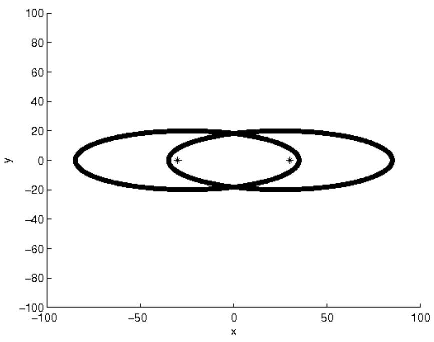
(b) B



(c) C

Figure 11: (A) shows 2 ellipsoids that intersect at exactly three points. (B) shows one degenerate conic, two intersecting lines, that passes through the three points of intersection. (C) shows a second degenerate conic, two parallel lines, that passes through the three points of intersection.

Euclidean Distance Approximation. A very naive approximation is to use the Euclidean distance between the centers of the ellipsoids. Figure 12 shows that this technique is prone to severe error.


(a) A

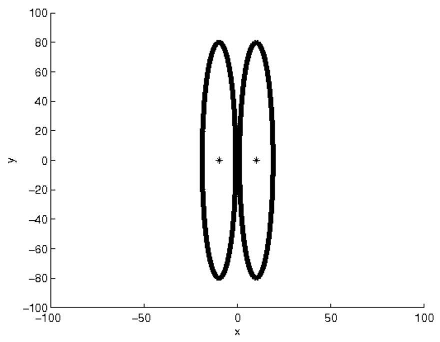

(b) B

Figure 12: (A) shows two ellipsoids whose centers are relatively far apart, yet they have significant overlap. (B) shows two ellipsoids whose centers are relatively close, but they have no overlap. These two examples show that distance between centers is not a good way to measure the proximity of two ellipsoids if their proximity is directly related to the amount of common area that they cover.

Hypersphere Estimation. One simple approximation of whether or not two ellipsoids e_1 and e_2 overlap is to simply find the longest semiaxis and then treat it as the radius of a circle. This simplifies the problem to determination of whether or not two hyper-spheres overlap. If the Euclidean distance from the center of e_1 to the center of e_2 is less than $r_1 + r_2$, where r_1, r_2 are the longest semiaxes of e_1 and e_2, respectively, then the two ellipsoids overlap. The problem with this method is that it is not accurate. Figure 12 shows two situations in which the weakness of this method is manifest.

Simple Overlap Check. In an ellipsoid, the distance from any point on the ellipsoid to the center of the ellipsoid is less than the length of the longest semiaxis and greater than or equal to the length of the shortest semiaxis (see Figure 13). Let ℓ_e^{max} and ℓ_e^{min} be the lengths of the maximum and minimum semiaxes, respectively, of hyper-ellipsoid e. Let $dist_{e_1,e_2}$ be the distance from the center of e_1 to the center of e_2. If $dist_{e_1,e_2} \leq \ell_{e_1}^{min} + \ell_{e_2}^{min}$, then e_1 and e_2 intersect [89]. If $dist_{e_1,e_2} > \ell_{e_1}^{max} + \ell_{e_2}^{max}$, then e_1 and e_2 intersect. If neither of these inequalities holds, then another method is necessary.

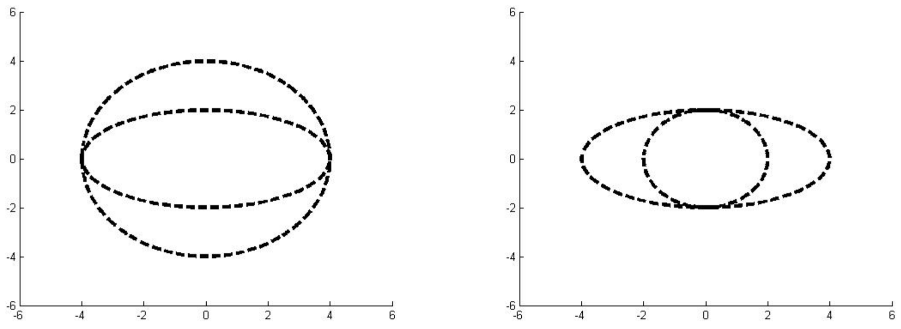


Figure 13: (a) shows an ellipsoid with an inscribed circle whose radius is equal to the length of the shortest semiaxis in the ellipsoid. (b) shows an ellipsoid with an circumscribed circle whose radius is equal to the length of the longest semiaxis in the ellipse.

Monte Carlo Method. We describe a statistical approximation using a Monte Carlo technique. In the following discussion, without loss of generality, we assume that the volume of e_1 is greater than or equal to the volume of e_2. In the Monte-Carlo technique, α n-d points are randomly generated inside of the hyper-rectangle rectangle which bounds e_1. This rectangle has the same center as e_1 and is oriented along the semiaxes of e_1. The rectangle has dimensions $2\ell_1, 2\ell_2, \ldots \ell_n$, $\ell_1, \ell_2, \ldots, \ell_n$ are the lengths of the semiaxes of e_1 (see Equation 16). Let h be the number of of the α points that fall inside of e_1 (see Section 3.2.3). Let *beta* be the number of points that fall inside of both e_1 and e_2. An estimate of the volume of overlap between e_1 and e_2 is

$$volume(rectangle) * \frac{h}{\alpha} * \frac{\beta}{h} = \left(\prod_{i=1}^{n} \ell_i \right) \frac{\beta}{\alpha} \tag{42}$$

The benefit of this technique is that it is approximate, so its computational complexity is significantly less than an analytical or precise numerical solution. Although this technique is approximate, it is statistically accurate. That is, the mean of the error over all overlap checks is 0. That is, there is an average of no error. However, the standard deviation of the error over all overlap checks is nonzero. The standard deviation of the error over all overlap checks is inversely related to α, the number of points that are randomly generated for sample checks. This error is acceptable as long as α is large enough so that the error is kept small.

Another note about this technique: Smith and Vamanamurthy [81] show that as n increases beyond 5 Ω_n, the volume of a unit ball in n-dimensions, rapidly approaches 0. This means that the number of randomly generated points that falls inside of the ellipsoid rapidly approaches 0 as n increases. This fact makes it difficult to generate random points inside of an ellipsoid using the bounding hyper-rectangle technique for dimension $n > 5$. One way to remedy this situation is to use a different bounding shape. A hyper-sphere whose radius is equal to the longest semiaxis in the ellipse could be used to bound the ellipsoid. If the ellipsoid is close to being spherically shaped, then the hyper-sphere could provide a tighter bound than the hyper-rectangle. However, if the ellipsoid has some semiaxes that are much longer than other semiaxes, then the volume of the bounding hyper-sphere could be significantly greater than the volume of the bounding hyper-rectangle. Hence, appropriate modifications are necessary when applying this technique for $n > 5$.

There are a number of possibilities for approximating the amount of overlap between two ellipsoids. Penalizing ellipsoids for overlapping other ellipsoids works well if the optimal solution does not include overlapping ellipsoids. However, we assume that a data set for which the optimal solution does not include overlapping ellipsoids is the exception, rather than the common case. To get an intuitive feel for why overlapping ellipsoids in an optimal solution is believed to be the common case rather than the exception, see the pedagogical data sets in Chapter IV. Figures 16, 17 and 18 show data sets in which the optimal solution is one or more non-overlapping ellipsoids. However, these data sets are specially designed to be such. Figures 19, 20 and 21 show data sets in which the optimal solution requires overlapping data sets. Penalizing ellipsoids for overlapping is not robust across all data sets, since most (all except special cases) data sets require overlapping in order to achieve an optimal solution.

Rewarding Only One Ellipsoid for Each Portion of Space

We investigate an inverse approach for the $REWARD$ function that does not include a penalty component. This approach rewards individuals that are far from others in the search space. Intuitively, this approach has more merit because it is more closely aligned with the goal of the problem. Again, the objective of the problem is to cover as much nonself space as possible with as few ellipsoids as possible. Penalizing ellipsoids for overlapping is based on the assumption that ellipsoids that overlap other ellipsoids are covering the same area twice, and thus probably not covering "new" area in the problem's n-space. However, as the previous section and Figures 19, 20 and 21 show, this assumption often leads to suboptimal performance. Rewarding only one ellipsoid for each portion of space covered only is closely aligned with the goal of the problem. If the reward function is based on the amount of area that is uniquely covered (or covered first-it is desired to award one ellipsoid for each area that is covered by at least one ellipsoid), then an increase in objective values indicates that the problem goal of covering more space is being better achieved. The same is not true when the $REWARD$ function includes a penalty for overlap with other ellipsoids.

In this approach, the objective values of the ellipsoids are evaluated by finding out which portions of the problem space are covered by one or more ellipsoids, and then rewarding only one ellipsoid for each portion of the problem's n-space that is covered. Solving this problem analytically requires calculating the sum of the volumes of each ellipsoid and then subtracting the overlaps that occur.

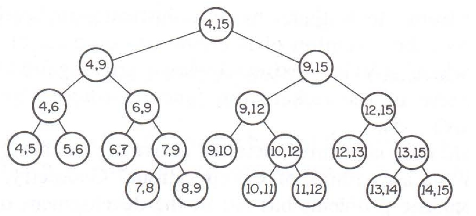
- Computing the sum of the volumes of the ellipsoids is simple: In fact, it is $O(n)$, where n is the dimensionality.
- "subtracting the overlaps that occur". This is very hard. It requires computing the area of overlap (see previous section) for every pair of ellipsoids and then doing complex calculations to achieve the desired result. This method is not investigated here because of its complexity.

We need an approximate method to decide which portions of n-space are covered by one or more ellipsoids and which ellipsoid should be given credit for each covered portion. To achieve this, we employ an n-way tree, which is the generalization of a segment tree.

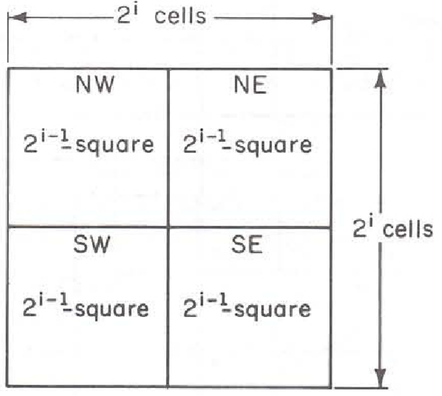
2^n-Way Tree. Preparata and Shamos [77, p.336-7] describe a data structure called the segment tree, which can be used for a wide variety of purposes. The term "segment tree" refers to an implementation in 1-d. An n-d implementation is referred to as a 2^n-way tree. i.e., in two dimensions, this data structure is referred to as a "four way tree."

One of the applications that Preparata and Shamos suggest is to answer the following question: Given a set of rectangles in some plane, how many of the rectangles contain a given point? When this question must be answered multiple times for the same (or very similar) set of rectangles, then using a n-way tree can save time over the naive approach of checking each rectangle against the point. Figure 14(a) shows an example of a one dimensional segment tree. Figure 14(b) shows an example of one node division in a $2d$ n-way tree, a 4-way tree, which is divided up into squares.

The basic idea is that the tree divides up n-space into hyper-rectangles. The leaves represent the smallest division. In Preparata and Shamos' application a subspace of the n-space is inserted into the tree by updating each node that represents a range that is contained in the subspace. In this way, each node contains a running count of how many inserted subspaces the node is totally inside of.



(a)



(b)

Figure 14: (A) shows the segment (4,15) inserted into a $1d$ segment tree. The nodes shown are those that the segment (4,15) covers. (B) Shows how one node is divided in a $2d$ segment tree.

An algorithm that uses a 2^n-way tree data structure to figure out how much space each ellipsoid uniquely covers is described below. A secondary goal for the solution of this problem is to use as few ellipsoids as possible. As a result of this, when more than one ellipsoid covers the same area in n-space, the largest of those ellipsoids is the only ellipsoid rewarded for covering that space. Rewarding the largest ellipsoids encourages solutions with larger ellipsoids. With larger ellipsoids, less ellipsoids are required to cover the space.

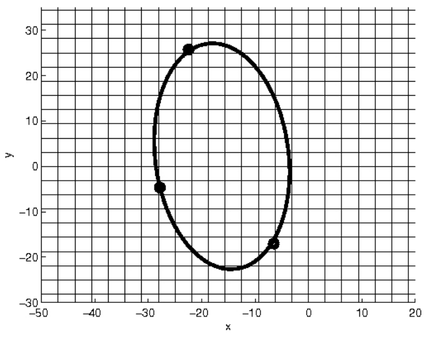
In order to compute an approximation for how much area an ellipsoid should be rewarded for, the ellipsoid is "inserted" into the 2^n-way tree.

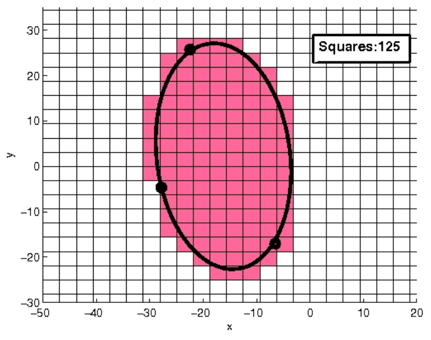
Inserting an Ellipsoid into a n-Way Tree

We investigate two approaches for inserting an ellipsoid into a 2^n-way tree: In the first, each ellipsoid is represented by a set of randomly generated points that are inside of the ellipsoid. These points are used to insert the ellipsoid into the tree. In the second method, an ellipsoid is inserted into the 2^n-way tree using several approximations to decide how much of each node is covered. Algorithm 1 describes the insertion of an ellipsoid using the first approach:

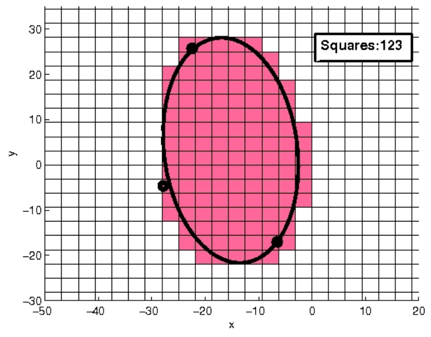
If the 2^n-way tree has τ levels, then the n-space is divided by the leaves into $2^{n\tau}$ hyper-rectangles. Each hyper-rectangle results in only one reward, for the first ellipsoid that covers that hyper-rectangle. Hence, if each ellipsoid is inserted into a 2^n-way tree that is originally empty (no ellipsoids) using the algorithm *insertEllipsoid*, then *insertEllipsoid* returns the fraction of the problem domain n-space that each ellipsoid covers uniquely, or for which it receives the reward for covering uniquely.

One of the weaknesses in this approach is that coverage accuracy is dependent on the granularity of the 2^n-way tree and the number of random points chosen to represent the ellipsoid. Sampling theory states that as the granularity of the 2^n-way tree increases, the number of random points must also increase in order to maintain coverage accuracy. If an ellipsoid has too few representing points in a granular 2^n-way tree, then the perceived area covered by the ellipsoid will be much smaller than the actual area covered by the ellipsoid. Also, this method does not perform equally for all ellipsoids because smaller ellipsoids need fewer representing points. Further, representing an ellipsoid by random points requires creating the points, which requires actually generating more points than the number desired, because some of the points will not fall inside of the ellipsoid. If the points are generated to be within a hyper-rectangle that bounds the ellipsoid, the bounding hyper-rectangle must also be computed. This is also not a trivial task. We do not use this method because of the overhead of generating points and bounding hyper-rectangles.


(a)



(b)



(c)

Figure 15: (a) shows an ellipsoid and three points that are not inside of the ellipsoid. The ellipsoid can cover more space, but only if it is mutated so that it is higher and further right. That mutation must be a center mutation. Any reorientation mutation causes the ellipsoid to cover one of the three self points, so that is not allowed. The semiaxis lengths cannot be mutated because any lengthening causes one of the self points to be covered, while any shortening causes the coverage of the ellipsoid to be decreased, so it is disallowed. Hence, the mutation must be a shift of the center of the ellipsoid to the upper right. (b) shows that, using a grid approximation for the coverage of the ellipse, 125 squares are covered (a square is considered "covered" if at least one of its corners is in the ellipse). However, (c) shows a small shift to the upper right that for which only 123 squares in the grid are covered. Analytically, however, the coverage of the ellipse is exactly the same because its size has not changed. If mutations that lower the reward function are always discarded in favor of the unmutated ellipsoid, it may be hard to find an acceptable center mutation toward the upper right.

The second method of inserting ellipsoids into the 2^n-way tree is described by algorithm 2. In Algorithm 2, each node in the 2^n-way tree maintains a value $c \in [0.0, 1.0]$ that represents the fraction of the node that has not been covered by previously inserted ellipsoids. Several approximations are used to decide how much of a node a hyper-rectangle covers. These approximations are based on the number of the corners of a 2^n-way tree node that are inside of an ellipsoid. Also, whether or not the center of the 2^n-way tree node is inside of an ellipsoid is used in the approximations. If $c = 1.00$ or an ellipsoid does not overlap a node at all, the ellipsoid receives no reward and returns to the node's parent. If an ellipsoid covers all of a node, it receives a reward and c is updated to 1.00. Otherwise, the ellipsoid traverses all children of a node and then returns to the node's parent. The 2^n-way tree insertion algorithm assigns a reward to each ellipsoid inserted. Although the 2^n-way tree algorithm is approximate, it is statistically successful in fulfilling the design goals of the reward function.

This method also has disadvantages. There are situations when an ellipsoid needs to have a center mutation. The center mutation is necessary because the ellipsoid changes it axis lengths or be rotated without lowering its objective value. The lower objective value may result from covering a self point or from covering less area. In some of these situations, the ellipsoid may have found an "optimal" covering of leaf nodes, that is, it is in a position where any small movement (such as a mutation), makes it lose objective value because it no longer touches the same number of leaf nodes in the 2^n-way tree. An example of this is shown in Figure 15. This risk is mitigated by using a fine grained (enough levels to produce so that the leaves cover relatively small areas of n-space) 2^n-way tree and by allowing mutations that are large enough to "jump" over the above described "dead space". We also mitigate this risk by using an approximate comparison operator.

Since a 2^n-way tree approximates the area that an ellipsoid covers, it is inappropriate to use an exact comparison operator when selecting ellipsoids for the next generation. An exact comparison operator is inappropriate because it is biased toward some ellipsoids and against some others. We apply an approximate comparison operator that uses a Gaussian distribution so that there is a nonzero probability of choosing an ellipsoid with a lower objective value as a better individual.

We choose the second *insertEllipsoid* algorithm (see Algorithm 2 in Appendix C) because it avoids the computation of generating random points and bounding rectangles. Also, every ellipsoid is treated similarly with the second approach, but with the first approach every ellipsoid may require a different number of random points. Hence, the reward function is simply the output of *insertEllipsoid* from Algorithm 2.

$$REWARD(E, e) = insertEllipsoid(e) \tag{43}$$

Penalty Function. The penalty function penalizes ellipsoids for covering self points. It is worse to cover more self points than less self points, so we desire to penalize more when an ellipsoid covers more self points. For this reason, it is necessary to count the number of self points that an ellipsoid covers. The naive solution to this problem is to simply check the ellipsoid against every self point using the Mahalanobis distance (see Equation 21). However, this quickly becomes a computational bottleneck as the number of self points increases. Thus, a more efficient technique is needed. We describe such a technique that makes use of the previously described 2^n-way tree.

Let S be the set of self points and let e be an ellipsoid. We desire to know how many points in S are inside of e. The 2^n-way tree tessellates n-space into 2^{ng} equally sized hyper-rectangles, where g is the number of levels in the 2^n-way tree.

Let $h(\alpha)$ be the hyper-rectangle represented by node α. Every $s \in S$ is inserted into the 2^n-way tree so that it resides in a leaf node α, whose hyper-rectangle $h(\alpha)$ contains p. Each node α in the 2^n-way tree maintains a counter, $count_\alpha$ that represents the number of self points inside $h(\alpha)$. Whenever $s \in S$ traverses node α while being inserted into the 2^n-way tree, $count_\alpha$ is incremented.

To find out how many self points are inside of ellipsoid e, our algorithm traverses the 2^n-way tree. At each node α, one of three things happens:

- If the algorithm can establish that e contains all of $h(\alpha)$, $count_\alpha$ is returned to α's parent. The algorithm can know that e contains all of $h(\alpha)$ if all of α's corner points are inside of e (see Mahalanobis distance in Equation 21).

- If the algorithm can establish that e contains none of $h(\alpha)$, 0 is returned to α's parent. The algorithm can know that e and $h(\alpha)$ do not overlap at all if the Euclidean distance between the centers of e and α is greater than the sum of the longest semiaxis length of e and the distance from the center to a corner in α.
- If the algorithm cannot establish either of the two above conditions, then all of α's children are traversed. If α is a leaf node, then the number of points in α that are inside of e is returned (see Mahalanobis distance in Equation 21).

s g represents a tradeoff. A higher value for g results in more hyper-rectangle nodes in the 2^n-way tree. If g is higher, then fewer of the self points must be checked because the approximate checks for containment and non-containment are more accurate. However, as g increases, more computation must be spent to compute the approximation checks. A smaller g means that more of the points must be checked, but less computation is spent computing the containment approximations at each node. We set the value for g based on the dimensionality of a problem. As the dimensionality increases, g must be decreased because memory consumption by the 2^n-way tree becomes too high.

If an ellipsoid e covers β self points, its penalty function is

$$PENALTY(e) = 1.00 - (REWARD(E, e)/(2^{\beta} + 1)). \tag{44}$$

With the reward and penalty functions defined, we define the objective function. $OBJECTIVE : \{E | E \text{ is a set of ellipsoids}\} \times E \rightarrow [0, 1]$

$$OBJECTIVE(E, e) = REWARD(E, e) - PENALTY(e). \tag{45}$$

3.3.4 Convergence. As the number of generations approaches infinity, the solution provided by the solution should approach some stable state. If not, it is impossible to know when the EA is finished, unlikely to find a better solution. This EA converges when the amount of space covered ceases to increase with additional generations. Several metrics for convergence are examined:

- Let χ be the sum of the fitness values of all individuals in the population. Convergence occurs as χ increases over generations. The algorithm has fully converged if $\chi_t = \chi_{t+1}$, where t is some epoch (generation number) and $t+1$ is one epoch later. The number of generations between t and $t+1$ should be large enough that χ has a good chance of increasing if it is not yet optimal (or close to optimal).

- If there are ϵ individuals in the population, let f_t^{ϕ} be the fitness of individual ϕ at epoch t. Also, assume that if ϕ_i, $1 \leq i \leq p$ are the individuals in the population, then they are sorted so that $f_t^{\phi_j} \leq f_t^{\phi_{j+1}}$ for $1 \leq j \leq \epsilon - 1$. Let $\chi = \sum_{i=1}^{\epsilon} \max_{\text{all generations in epoch } t}(f_t^{\phi_i}) - \max_{\text{all generations in epoch } t-1}(f_{t-1}^{\phi_i})$ In this technique the sign that convergence is the decrease of χ. Convergence has fully occurred when $\chi = 0$. The general idea is that convergence occurs when the n_{th} best individual, $1 \leq n \leq \epsilon$, in a population is not getting any better as generations pass. This technique is chosen over the first technique described because the value that χ is approaching for convergence is known, since it is 0. In the first technique, however, the value that χ is approaching is not known, since it is a function of the optimal solution, and the optimal solution (or even fitness of the optimal solution) is not known.

In an EA, it is important to explore the solution space early and then exploit good solutions as convergence approaches. We implement this behavior by changing the value of the standard deviation used for the Gaussian distributions in ellipsoid mutation. Decreasing the standard deviation for mutation operators is a technique from simulated annealing [62]. Simulated annealing is inspired by the physical process of cooling a material. In this process, a material is slowly cooled so that at each temperature the material is provided with sufficient time to settle into a intermediate low energy state.

We implement a simulated annealing technique by initializing the mutation standard deviation to a high value so that ellipsoid mutation explores the solution space. As the algorithm progresses and completes more generations, the mutation standard deviation is decreased. This allows the algorithm to exploit the good solutions that it has already found. The ellipsoids "settle" into a final converged solution. Eventually, the standard deviation approaches 0. When the standard deviation is sufficiently close to 0, the ellipsoids cease to become better through mutation. This is convergence.

3.3.5 Algorithm Parameters. This section defines input parameters to our algorithm. These parameters are tuned based on the dimensionality of the data set, desired accuracy of results, and desired time to obtain results. Also, they can be "tuned" to increase performance depending on individual characteristics of data sets.

- **Population Size**: The number of detectors that are perpetuated from generation to generation. A higher number means that detectors cover more space, although too many detectors could result in detectors covering small holes in self space.
- **Number of Children**: The number of children that are randomly generated at each generation. A higher number results in more exploration. The tradeoff is that a higher number of children means more computation.
- **Maximum Generations**: The algorithm quits after this number of generations.
- **Generations Between Convergence Check**: This value is the number of iterations between a convergence check (see Section 3.3.4). If the convergence check finds that no detectors in the population have improved since the last convergence check, then the mutation standard deviation is decreased by a factor of 0.10. If this value is too high, the algorithm takes an unnecessarily long time to converge. If this value is too low, the algorithm may converge too quickly to a local optimum.
- **Initial Mutation Standard Deviation**: The initial standard deviation for mutation.
- **2^n-way Tree Depth**: The depth of the 2^n-way tree described in Section 3.3.3. A higher number results in greater accuracy for ellipsoid approximations. However, a higher number also results in significantly increased memory for the data structure and increased computation to traverse the tree..
- **Choose Worse Detector Standard Deviation**: This is a standard deviation used for a Gaussian distribution in the approximate comparison operator. A higher number means that the comparison operator is more likely to choose a worse detector.

- **Settle Phase Begin Generation**: This value is a number of generations. After the specified generation has passed, a worse (objective value) detector is can only be chosen over its unmutated original if its volume is greater. This encourages convergence. If this value is too high, the algorithm could take an unnecessarily large amount of generations to converge. However, if this value is too small, this algorithm could converge prematurely to a local optimum.
- **Dimensions**: The number of dimensions in the data set.
- **Geometry Type**: This specifies whether ellipsoids or spheres are to be used. Specifying sphere means that rotation mutation is disabled and all semiaxis lengths are mutated identically.

The described designs for mutation, objective function and convergence, along with the model for ellipsoids, which are the individuals in the EA, define an algorithm and data structures that should be able to produce a set of ellipsoids that cover nonself space and cover as few self points as possible.

3.4 Algorithm Summary and Complexity

This section provides pseudocode for the whole algorithm. The time computational complexity of generating ellipsoids using this algorithm is $O(2^{n+n\alpha} * n^2 * genMax)$, where n is the number of dimensions, α is the depth of the 2^n-way tree, and $genMax$ is the maximum number of generations for the EA. The space complexity is $O(popSize + 2^n * 2^{n\alpha}) = O(2^{n+n\alpha})$, where n is the number of dimensions and α is the depth of the 2^n-way tree. The reason for the space complexity is that each node in the 2^n-way tree maintains a list of its 2^n corner points.

3.5 Implementation Details

This section describes the justification for lower level implementation decisions.

3.5.1 Feature Representation: Binary v. Real Value. This section also deals with the definition of the $findMalicious$ function in Equation 8 (finds points of class self. Once a model has been chosen (*toData* function in Equation 5 converts raw data to n-d data for input to classification system), a decision must be made about how to represent the feature values in a computer. Most features in a selected network model are represented by some type of numeric value, or a group of numeric values. Of course, the question arises, "Which is better?". Obviously, computers handle binary computations, so this would seem to be a "natural" represention given the desire to use computers to solve the current problem. However, some argue that using binary values makes it difficult to interpret results. For instance, if two data points "match" using some binary matching function, what does that mean? It could be that this "match" is an artifact of the binary matching function, and it may be difficult to interpret the binary values anyway. If real values are used, however, it is more intuitive to interpret the meaning when two data points match according to some real valued matching function.

We use real values because it is intuitive and easy to interpret. This is especially true because we are doing geometric operations in Euclidean space.

3.5.2 Implementation Language. Because of library and performance issues, we initially narrow our programming language choices to Java and C/C++. There are several factors to analyze when choosing a programming language for implementation of our algorithm:

- Performance Requirements-our algorithm does require significant computation, so performance is important. However, since we do not desire to measure execution time and since we are not attempting to build a release-quality product performance is not the only factor in our decision. Generally, native code has the best performance, assuming a reasonable compiler. Compiled C/C++ is usually assumed to achieve lowest execution runtime. Java code is not native, but it is also not interpreted at the source code level like Perl or Scheme. Java compilers generate byte code, which is slower than native code but faster than high level interpreted languages like Perl. Hence, although Java programs usually run slower than C/C++ programs, it is only a constant slowdown, and not orders of magnitude.

- Portability-this is Java's strength. Java byte code be run on any platform that has the JRE, the Java Runtime Environment. There is a JRE for just about all platforms. C/C++ however, has to be recompiled for each platform. Recompilation can be a problem if libraries are not the same on different platforms or if the portions of the C/C++ code are platform specific.
- Existing Libraries-since our algorithm implements an evolutionary algorithm and does Matrix manipulation, it is important to find good libraries for these tasks. Many libraries exist in C/C++ for EAs and matrix operations. Java libraries for EAs and matrix operations (see [3], [1]) exist, but are fewer. However, we did find good Java libraries: JosephGA (EA library written by Joseph Shapiro at AFIT) and JAMA (Java MAtrix Package) [2]. Another benefit of Java is a very complete standard library, which results from the fact that Java is maintained by Sun, whereas C/C++ does not have a single owner.
- Although C/C++ and Java are both high level languages, Java offers more abstracted constructs, is more structured, and more strongly typed. For this reason, Java is easier to learn than C/C++.
- Software Engineering Model-it is important to use an object oriented software engineering model because if makes a program modular brings the benefits of inheritance and polymorphic behavior. Both Java and C++ offer an object oriented paradigm, although Java enforces it while most C++ compilers allow reversions to C, which is not object oriented. Another benefit of Java is the Javadoc documentation builder, which works very well and is used almost universally for documenting Java programs.

  We choose Java for the following reasons:

- We are very familiar with JosephGA, an EA package for Java
- Java's portability is important. Statistically sound testing requires that many runs be made against each data set. Portability allows for development in a convenient Windows environment and large scale testing on Linux clusters.
- The Java standard libraries are good, from a centralized source (Sun) and well documented.

- The performance benefits gained by using C/C++ are not necessary since we are not timing runs for performance analysis and we do not desire this product to be production quality. If a production quality product using ideas from this research is desired at a later date, the algorithm can be ported to C/C++ with a reasonable effort.

The next chapter describes testing procedures that can be used to evaluate how well this algorithm and data structures work to evolve ellipsoids to cover nonself space.

IV. Experimental Design

The objective of this section is to design tests that can test how well the algorithm in this research works. The tests are designed to answer the following questions:

- Validation of model and algorithm: Does an evolutionary algorithm that mutates ellipsoids evolve a good set of ellipsoids to fill nonself space? We answer this question qualitatively using pedagogical 2d data sets. These pedagogical data sets are artificially produced. This is convenient for validation because data sets can be produced for which an optimal solution is known. Also, final solutions and even the evolution of solutions can be graphically visualized in ways that can validate the effectiveness of the algorithm and provide insight into the algorithm's behavior. Quantitative analysis is accomplished for the pedagogical data sets through analysis of the percentage of test points correctly classified. For the real world data sets, quantitative analysis is accomplished by measuring true positive and false positive rates. Also, our results are compared with the results of others on the same data sets.
- Does the evolutionary algorithm converge? Graphs of solution performance by generation show the convergence properties of our algorithm.
- How does the ellipsoid method compare with other geometric shapes? We hypothesize that the performance of ellipsoids is better than spheres. Do results validate this hypothesis? This question is answered qualitatively through visualizations of 2d pedagogical data sets. To answer this question quantitatively, we measure performance of ellipsoids and spheres against identical data sets and compare this performance.
- Are results data set dependent? Tests are performed for several data sets, thus enabling comparison between different data sets.

The above questions are first answered on pedagogical problems to validate the model. Then, real world data sets are used.

Since this algorithm is stochastic, statistical analysis is required for analysis. If we assume that results follow a Gaussian distribution, a higher number of runs achieves a higher confidence interval. However, since the algorithm is not trivial (time and space complexity) to run, there must be a tradeoff between the confidence interval and the number of runs. We choose to use 30 runs, which is a good tradeoff. 30 runs provides a relatively tight confidence interval but is still computationally feasible.

Also, this chapter describes a suggested taxonomy for data sets that can be used for negative selection.

4.1 Validation of Model

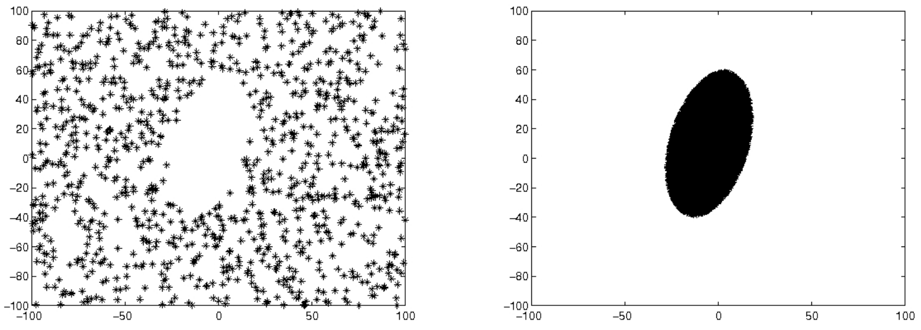
4.1.1 Pedagogical Problems. Smaller problems are important because they are a good proof of concept. Also, it is easier to validate that an algorithm is producing expected results on pedagogical problems because much information about the problem is known (because it is probably hand-produced). Another benefit of pedagogical problems is that they can be smaller and lower dimension so that intuitive visualization techniques can be used.

The first problems investigated are artificially produced data sets in which the geometry is elliptical or inverted elliptical. In four of these data sets, nonself space is elliptically shaped, so the optimal solutions (ellipsoids that fit the nonself space) are known a priori. Since the optimal solutions are known, it is easy to decide see whether or not the algorithm finds the best solution. Figures 16 - 18 and 21 represent this type of problem. Figures 19 and 20 are inverse problems. They test how well the algorithm can find a set of ellipsoids to fill in a space that is not elliptically shaped.

Each of these data sets consists of 1000 self points (for training) and 10000 nonself points (for testing). No self points are used for testing because the ellipsoids produced by our algorithm totally avoid self points, so they are unlikely to cover any during the test phase. The numbers 1000 and 10000 are chosen somewhat arbitrarily. There is a tradeoff between higher and lower numbers of self and nonself points. A larger number of self points means that ellipsoids are more likely to find the optimal ellipsoid solution (against data sets for which the optimal solution is an ellipsoid), since the self area is more precisely defined during training. A larger number of nonself points results in more accurate testing, since the nonself area is more precisely defined for testing.

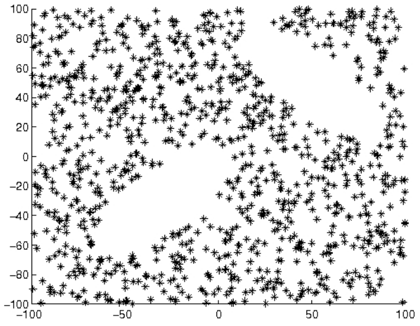
For convenience' sake these pedagogical data sets are assigned abbreviated names PD1-PD6, where PDn is the n^{th} pedagogical data set. Short descriptions of these data sets are provided:

- **PD1**: 2d data set with one ellipsoidally shaped hole (See Figure 16). It is expected that one ellipse covers this hole very well.
- **PD2**: 2d data set with two ellipsoidally shaped holes of the same size, but oriented differently(See Figure 17).
- **PD3**: 2d data set with two ellipsoidally shaped holes of different size, oriented differently (See Figure 18).
- **PD4**: 2d data set with one ellipsoid that should not be covered (See Figure 19). This problem is interesting because ellipses in the solution must overlap.
- **PD5**: 2d data set with two ellipsoids that should not be covered (See Figure 20). Again, this problem is interesting because ellipses in the solution must overlap.
- **PD6**: 2d data set with two ellipsoids that form a cross shape. (See Figure 21). Unlike the data sets in Figures 19 and 20, the optimal solution for this problem is known. The optimal solution is two crisscrossing ellipses. This data set tests not only if the algorithm can find a good solution, but also whether or not the algorithm can find the optimal solution.



(a) a (b) b

Figure 16: (a) is a data set with one elliptical hole. (b) is its associated test data set.


(a) a

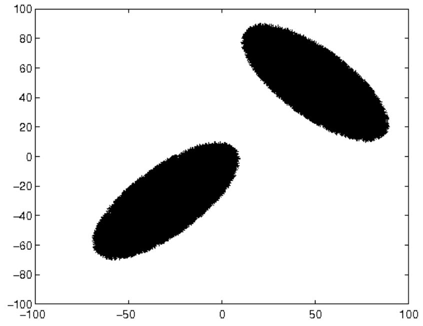
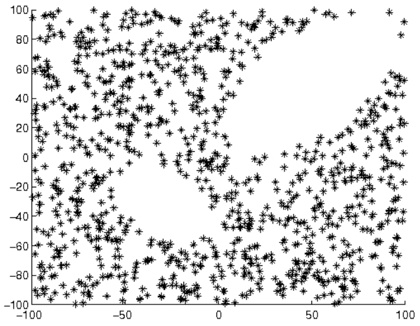

(b) b

Figure 17: (a) is a data set with with two elliptical holes. The ellipsoids are the same size, but are oriented differently. (b) is its associated test data set.


(a) a

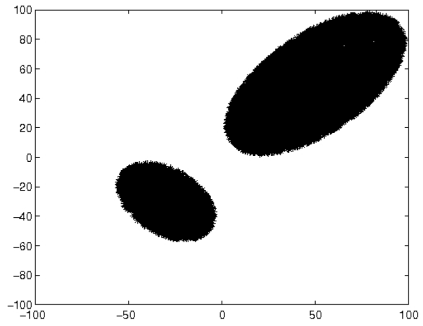
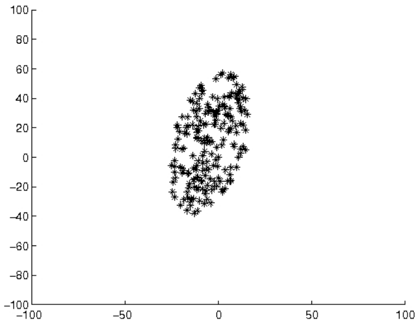

(b) b

Figure 18: (a) is a data set with with two elliptical holes. The ellipsoids are oriented differently and are different sizes. (b) is its associated test data set.


(a) a

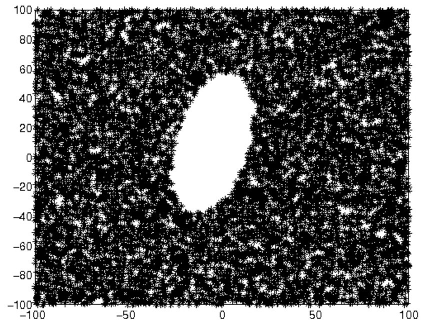
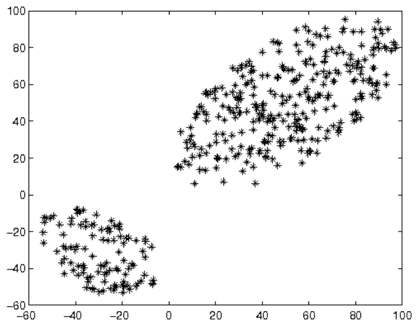
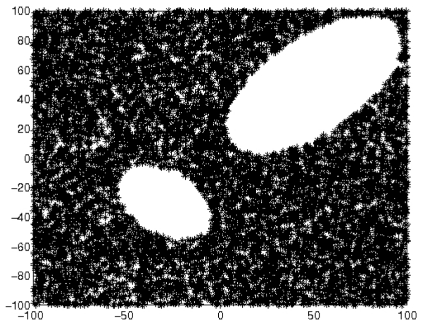

(b) b

Figure 19: (a) has points inside of an ellipsoid, instead of outside of the ellipsoid. (b) is its associated test data set.

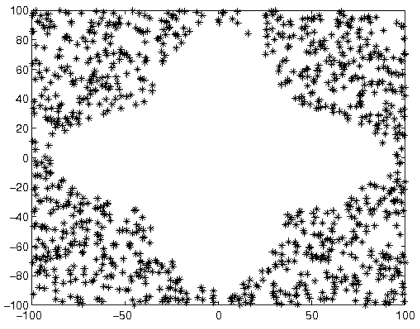


(a) a

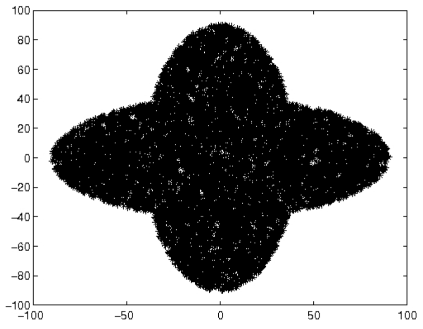


(b) b

Figure 20: (a) has points inside of two ellipsoids. (b) is its associated test data set.



(a) a



(b) b

Figure 21: In (a), the optimal solution is obviously two ellipsoids in a cross formation. (b) is its associated test data set.

The algorithm is run 30 times for each data set with each geometry type (ellipsoid or sphere) to provide results that can be analyzed with first (mean) and second (standard deviation) order statistics. This means that there are 6 [data sets] $\times 2$ [geometry types] $\times 30$ [statistics] = 360 total runs against the pedagogical data sets. Parameters for each run are shown in Table 5.

Table 5: Parameters for runs against Pedagogical data sets. "E"is for Ellipsoid, "S" is for Sphere, "D1" means data set 1, etc.

	D1		**D2**		**D3**		**D4**		**D5**		**D6**	
Parameter	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**
Population Size	1	15	2	20	2	20	12	20	20	20	12	20
Number of Children	1	1	1	1	1	1	2	2	2	2	2	2
Maximum Generations (x 1000)	20	10	20	10	20	10	20	10	20	10	20	10
Detector Count	1	15	2	20	2	20	12	20	20	20	12	20
Generations Between Convergence Check	500	100	500	100	500	100	500	100	500	100	500	100
Mutation Standard Deviation	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
Choose Worse Detector Standard Deviation	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Segment Tree Depth	4	4	4	4	4	4	4	4	4	4	4	4
Generations Between Classification Check	50	50	50	50	50	50	50	50	50	50	50	50
Settle Phase Begin Generation (x 1000)	15	7.5	15	7.5	15	7.5	15	7.5	15	7.5	15	7.5

After the algorithm has been run against the data sets in Figures 16 - 21, a set of nonself points are needed for testing. Test data is obtained by generating random points in the inverse of the self area in each data set. A large number of points is desired so that test results accurately reflect the performance of obtained solutions. 10,000 is chosen as the number of nonself points to test against. Part (b) of Figures 16 - 21 shows that 10,000 points provides good coverage of the nonself region.

4.1.2 Real World Test Data Sets. For real world validation, we use established classification data sets for comparison purposes and the DARPA IDS evaluation data to validate our algorithm in the intrusion detection domain. Unfortunately, the negative selection research community has not established a suite of data sets which are believed to be good for testing negative selection algorithms. In fact, there is not even a set of data sets which are commonly used by multiple research efforts for comparison purposes. The only data set that is common among negative selection research is the 1999 DARPA Lincoln Labs IDS data. [61]. A handful of other data sets have been used, but generally not more than once, even in the progression of the same research effort. These data sets include:

- random binary strings [34]
- SPARC instructions generated by compiling C programs. Anomalies created by making changes in source code and recompiling. [34]
- COM files infected with artificially inserted viruses. [34]
- tool breakage detection [49]
- time-series anomaly detection [38]
- biomedical data [49]

We choose real world data sets with varying characteristics to validate that our algorithm works against data sets with a varying range of characteristics. We base our choice test data sets on the characteristics of number of classes, dimensionality, and size of data set. When there are more than two classes, one class is used for self and the complementary classes are used for nonself. Hence, nonself can be a multi-modal class as a result of conglomerating two or more problem classes.

Iris data set: We obtain the Iris data set from the University of California at Irvine Machine Learning Repository [18]. The iris database originates from a classic taxonomy paper describing three different types of iris plants: Iris-Setosa, Iris-Versicolor, and Iris-Virginica. Each data base instance represents a plant belonging to one of these three classes. The database contains 50 instances of each class, for a total of 150 instances. Each instance has four real-valued attributes corresponding to the plant's sepal length, sepal width, petal length, and petal width. The lengths are measured in centimeters. Of the three classes, Iris-Setosa is linearly separable from the other two, but the other two are not linearly separable from each other.

The iris data set is tested using a 90/10 test. With a 90/10 test one class is chosen as self for training. 90% of the chosen class is used for training. The other 10% of the chosen class, along with 10% of each of the other two classes is then used for testing. Ten different sets are used, each time using a different 10% of each class for testing. For statistical purposes, each unique test is performed 10 times. Tests are performed using each of the three classes as the self class. This means that there are 3 [classes] $\times 10$ [10% subsets] $\times 10$ [statistics] = 300 total runs against the Iris data set. Table 6 shows algorithm parameters for the Iris data set.

Lincoln Labs Data Intrusion Detection Data: This data is from the 1999 DARPA IDS Evaluation Data Set [61]. For training, we use the week one data, which contains only normal traffic. For testing, we use week two data, which consists of normal traffic mixed with attacks. The data has three features: number of bytes per second, number of packets per second, and number of Internet Control Management Protocol (ICMP) packets per second. The data we use focuses on outside network attacks using data filtered with the program *tcpdump* [9]. The same data set is used by Dasgupta et al. [24] [38] and Mills et al. [67]. The attacks in this data set are shown in Table 8. Back and Neptune are denial-or-service (DOS) attacks, while Portsweep and SATAN are stealthy attempts to discover weaknesses in the network security perimeter.

Table 6: Parameters for runs against Iris data sets. "E"is for Ellipsoid, "S" is for Sphere

	Train Setosa		**Train Versicolor**		**Train Virginica**	
Parameter	**E**	**S**	**E**	**S**	**E**	**S**
Population Size	2	25	25	50	30	50
Number of Children	1	1	2	2	2	2
Maximum Generations (x 1000)	1	1	1.5	2	1.5	2
Detector Count	2	25	25	50	30	50
Generations Between Convergence Check	500	500	500	500	500	500
Mutation Standard Deviation	0.15	0.15	0.15	0.15	0.15	0.15
Choose Worse Detector Standard Deviation	0.15	0.15	0.15	0.15	0.15	0.15
Segment Tree Depth	3	3	3	3	3	3
Generations Between Classification Check	50	50	50	50	50	50
Settle Phase Begin Generation (x 1000)	1	1	1.5	2	1.5	2

Table 7: Normalizing MIT IDS Data

dimension	min	max	min used for normalizing	max used for normalizing
1	0.00	15835.03	-16000	16000
2	0.00	338.08	-400	400
3	0.00	12.233	-14.92	14.92

The MIT Lincoln Labs data is preprocessed for normalization. Normalizing is important in our algorithm because of orientation mutation. If the ranges for different dimensions vary significantly, there are likely to be ellipsoids that are very long in one dimension but very short in another. This makes finding a valid rotation almost impossible because, unless the rotation is very small (VERY SMALL), the rotation causes the ellipsoid to cover a self point, which leads to rejection of the mutation and the unmutated ellipsoid is retained in the population. Hence an ellipsoid that needs a rotation to become more optimal may never get that rotation.

One way of dealing with this issue is to choose a rotation angle based on the relative dimension ranges. However, this method is fraught with complications because a semiaxis may not be aligned along a coordinate axis and it adds unnecessary complexity to the algorithm. We choose to deal with this issue by requiring data sets to be normalized so that the bounds for reach dimension are [-100.00, 100.00]. The mapping from original bounds for each feature to normalized bounds for each feature is somewhat arbitrary. For our experiments, we perform this mapping by hand. The guidelines we use are that all data points should be inside the mapped range and there should be at least 50% of each range that does not contain any data points. This 50% should be approximately equally divided above and below the most extreme points in each dimension. The "padding" gives the ellipsoids room to work with the constraint that the center point of each ellipsoid must be inside of [-100.00, 100.00], the problem domain bounds. Table 7 shows how the MIT ID data is normalized.

Table 8: Week two attack profile. [67]

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

Table 9: Parameters for runs against MIT IDS data. "E"is for Ellipsoid, "S" is for Sphere

	Train Setosa		Train Versicolor		Train Virginica	
Parameter	**E**	**S**	**E**	**S**	**E**	**S**
Population Size	2	25	25	50	30	50
Number of Children	1	1	2	2	2	2
Maximum Generations (x 1000)	1	1	1.5	2	1.5	2
Detector Count	2	25	25	50	30	50
Generations Between Convergence Check	500	500	500	500	500	500
Mutation Standard Deviation	0.15	0.15	0.15	0.15	0.15	0.15
Choose Worse Detector Standard Deviation	0.15	0.15	0.15	0.15	0.15	0.15
Segment Tree Depth	3	3	3	3	3	3
Generations Between Classification Check	50	50	50	50	50	50
Settle Phase Begin Generation (x 1000)	1	1	1.5	2	1.5	2

4.2 Test Data Set Taxonomy

We seek to produce data sets that can be used to test the robustness of a negative selection algorithm and that can be used by various research efforts to compare negative selection algorithms. First, we identify important characteristics of a data set that is used to test a negative selection algorithm:

- Dimensionality.
- Data point density-this is important because it is much easier to train when there is a high density of data points. In intuitive terms, it is harder to easier to know with surety that a portion of space should be assigned a class when there are more points in that space (or in its complementary space.
- Geometric Shapes/Distributions. Not all detectors are good at detecting all geometric shapes. i.e., An ellipsoid detector may have trouble with hyper-rectangles and vice versa.
- Number of shapes. How many instances of each shape are in the data set?
- Are geometric shapes inverted? This is asking if the self class is a geometric shape or if the nonself class is a geometric shape. If the nonself class is a geometric shape, then the task of the detectors is to detect the complement of a geometric shape, a rather different problem.

This suggested taxonomy is not an attempt to define a system for classifying all negative selection test data sets. It is a first suggestion at a way this could be done. However, we do classify our test data sets (PD1-PD6) according to this taxonomy. Let data point density be the number of self train points per unit area in the problem domain bounds. For the inverted category, we let “no” mean that nonself space is shaped like the geometric shape and “yes” mean that nonself space is shaped like the complement of the geometric shape. The classification for data sets PD1-PD6 is shown in Table 10.

Table 10: Characteristics of test data sets.

Data Set	**Dimensions**	**Data Point Density**	**Shape**	**Number of Geometric Formations**	**Inversion**
1	2	0.100	ellipse	1	yes
2	2	0.100	ellipse	2	yes
3	2	0.100	ellipse	2	yes
4	2	0.100	ellipse	1	no
5	2	0.100	ellipse	2	no
6	2	0.100	ellipse	2	yes

The next chapter reports the results of the experiments described in this chapter. Analysis is performed to evaluate our algorithm's level of success.

V. Results and Analysis

The chapter presents and analyzes results from the experiments designed in Chapter IV. Sections 5.1, 5.2, and 5.3 analyze results from the pedagogical, MIT, and Iris data sets respectively.

5.1 Pedagogical Problems

This section addresses one of the questions set forth in Chapter IV, "Do results validate the usefulness of our model and algorithm?" True positive and false alarm rates show that our model has works well against some data sets.

The algorithm performs well on PD1-PD6, the pedagogical problems in Chapter IV. Figures 23 - 28 show that the algorithm generates ellipsoids that successfully fill the elliptically shaped holes. Figure 29 shows that the evolutionary algorithm converges to a set of ellipsoids such that most of the test points are covered, which is the goal. The algorithm is expected to perform well on the first three datasets because the algorithm is trying to fill disconnected elliptically shaped spaces. Figure 29 shows that, in the first three datasets, the entire elliptical holes are generated by the generated ellipsoids. In the fourth data set, the algorithm is trying to fill a space that is not elliptically shaped. The algorithm does find a set of ellipsoids that entirely covers the target space because there are enough ellipsoids that a good cover can be obtained. The fifth data set also contains a non-elliptical space that should be covered by generated ellipsoids. The generated ellipsoids are good, covering about 95% of the space. The space is not entirely covered because there are not enough ellipsoids. In the sixth data set, the obvious optimal solution is two ellipsoids. The algorithm finds two ellipsoids that cover 95% of the space. This case shows that the algorithm can find an optimal ellipsoid cover when the known optimal ellipsoid cover includes intersecting ellipsoids.

Results also show that the algorithm converges. It is important that the algorithm converge. If an evolutionary algorithm does not converge, then the solution it provides is a function of the generation at which the solution is chosen. Hence, the evolutionary algorithm is not really providing a solution, but rather a different solution at each generation.

Table 11: Ellipsoid, Sphere Performance Against PD1-PD6. "Count" is the number of detectors, "Average" is coverage over 30 runs, "Std Dev" is over 30 runs.

Data Set	**Ellipsoids**			**Spheres**		
	Count	**Average**	**Std Dev**	**Count**	**Average**	**Std Dev**
PD1	1	99.82	0.73	8	91.21	3.86
PD2	2	98.42	3.29	17	94.98	1.64
PD3	2	99.82	0.32	16	93.40	2.77
PD4	12	99.53	0.15	20	99.01	0.42
PD5	12	97.53	0.28	20	95.57	0.53
PD6	2	99.48	0.14	20	94.39	1.57

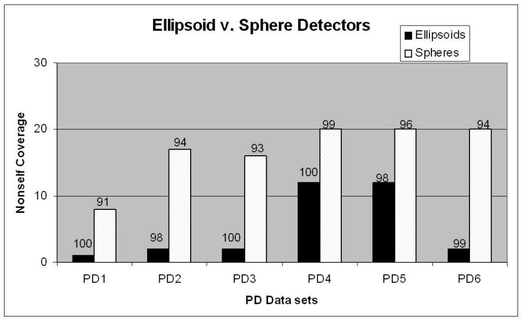


Figure 22: Comparison of Ellipsoid detector count v. Sphere detector count. The numbers above the columns in the bar graph are the percentages of nonself points that were covered by the detector described in the respective bar.

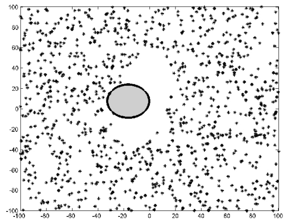
If there is a different solution at each generation, it is impossible so say that the algorithm provides good solutions. Hence, it is important to show that an algorithm converges if we desire to make statements about the quality of the solutions the algorithm produces.

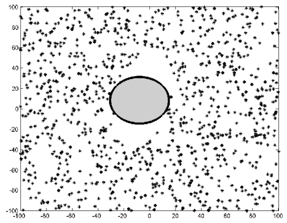
For a convergence metric we use the performance of the generated ellipsoids. We say that an algorithm “converges” if the performance of the generated ellipsoids stabilizes after a sufficient number of generations. Figure 29 shows that the performance of generated ellipsoids approaches a steady state as more generations are completed. The thicker line shows the fraction of the test points the ellipsoids cover after each generation. The upper thin line shows the value of the average fraction covered plus the standard deviation. The lower thin line shows the value of the average fraction covered minus the standard deviation. In all cases, the algorithm converges to a stable set of ellipsoids.

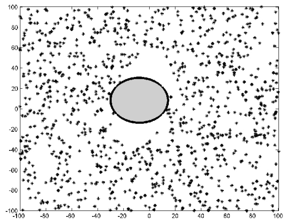
The fact that the graphs in Figure 29 are not smooth is a result of the segment tree data structure and the approximate comparison operator. These two algorithm components make it possible for the generated ellipsoids to become worse after a generation. However, as the graphs show, the algorithm statistically chooses the best ellipsoids, so the long term performance is good.

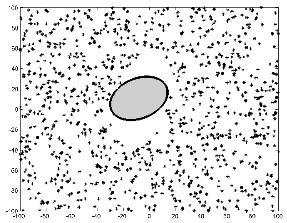
We hypothesize that spheres can cover space as well as ellipsoids, but that more spheres than ellipsoids must be used to achieve similar coverage of nonself space. Table 11 shows that this is the case. It takes more spheres than ellipsoids to cover the same amount of nonself space. Figure 31 shows how the amount of nonself space covered compares to the number of spheres.

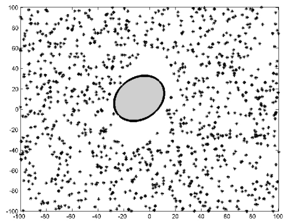
The graphs in Figure 29 all show a small bump at generation 15000. This bump is a result of the parameter “Settle Phase Begin Generation.” The bump happens because worsening mutations are not allowed after “Settle Phase Begin Generation”, so the ellipsoids quickly fill up whatever hole they happen to be in at “Settle Phase Begin Generation”. PD6 actually shows a small drop at “Settle Phase Begin Gen”. The reason for this is that there are 12 ellipsoids in the population, but only two are used for classification. Those two are not the best at “Settle Phase Begin Gen”, but they quickly grow to become the best.

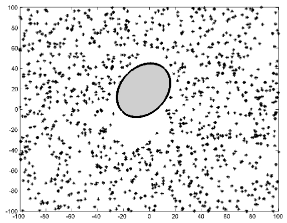

(a) 8 Generations

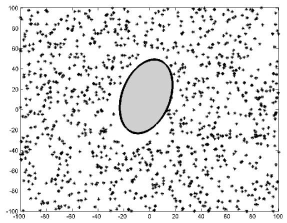

(b) 20 Generations

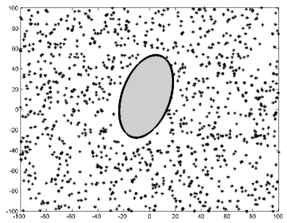

(c) 250 Generations

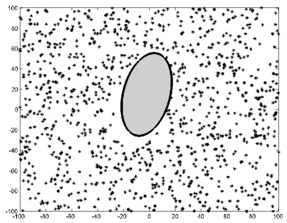

(d) 490 Generations

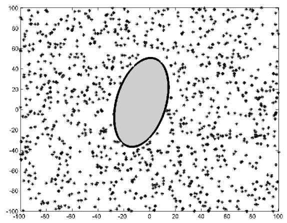

(e) 525 Generations

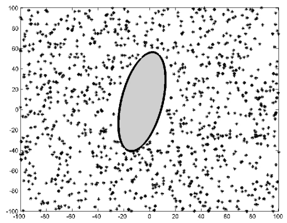

(f) 560 Generations


(g) 600 Generations


(h) 650 Generations


(i) 725 Generations


(j) 910 Generations


(k) 1250 Generations

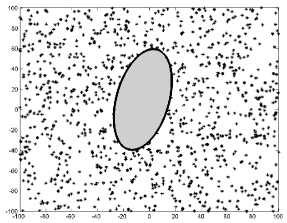
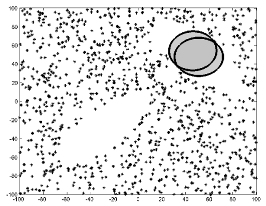
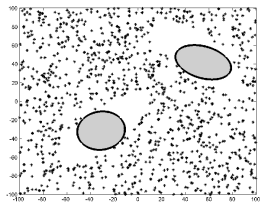
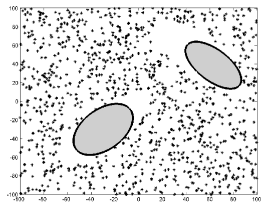
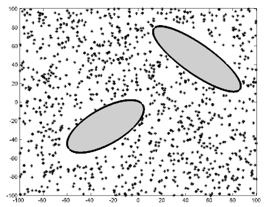

(l) 10000 Generations

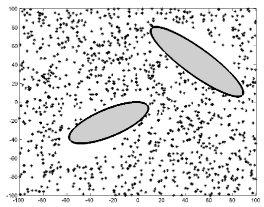
Figure 23: The Evolution of one ellipsoid using self data PD1.

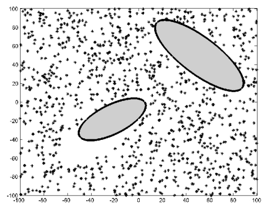

(a) 100 Generations

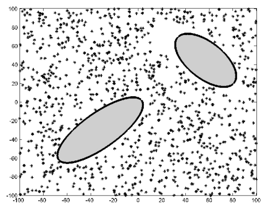

(b) 300 Generations

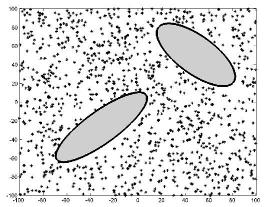

(c) 400 Generations

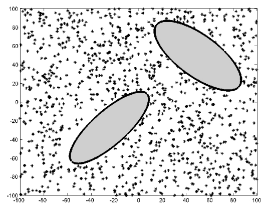

(d) 700 Generations

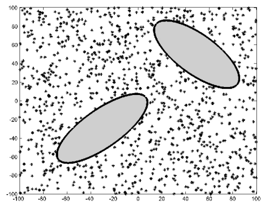

(e) 1200 Generations

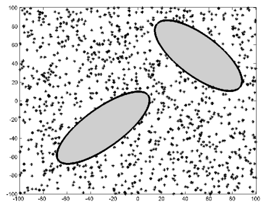

(f) 2000 Generations


(g) 2500 Generations


(h) 3500 Generations


(i) 4500 Generations


(j) 5500 Generations


(k) 7000 Generations

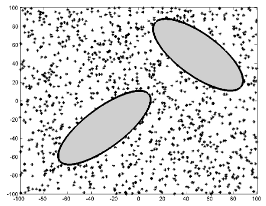
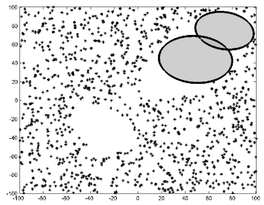
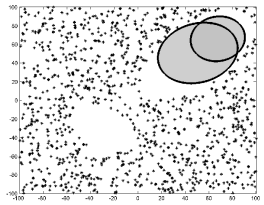
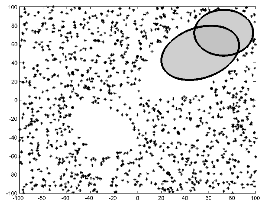
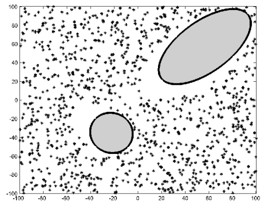

(l) 10000 Generations

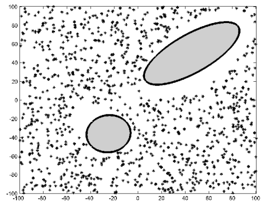
Figure 24: The Evolution of two ellipsoids using self data set PD2.

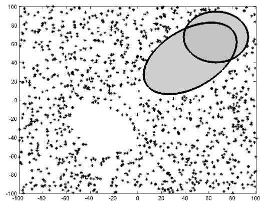

(a) 100 Generations

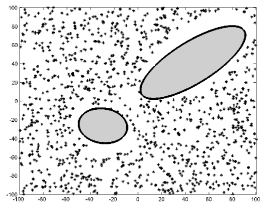

(b) 300 Generations

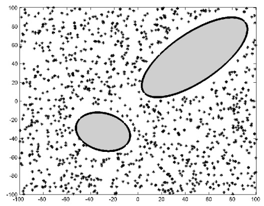

(c) 400 Generations

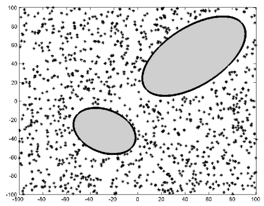

(d) 700 Generations

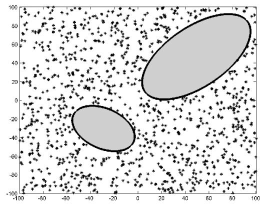

(e) 1000 Generations

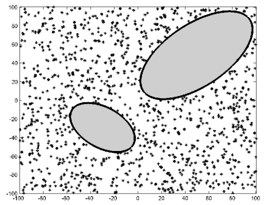

(f) 1600 Generations


(g) 2800 Generations


(h) 3000 Generations


(i) 3200 Generations


(j) 3500 Generations


(k) 5000 Generations

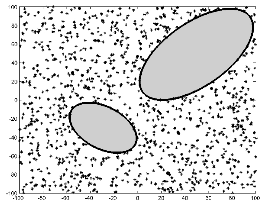
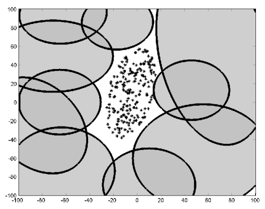
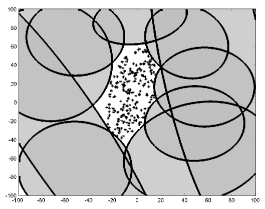
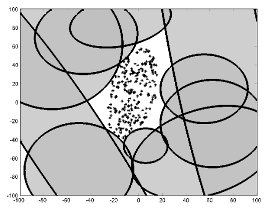
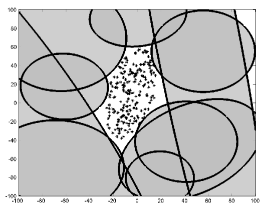

(l) 10000 Generations

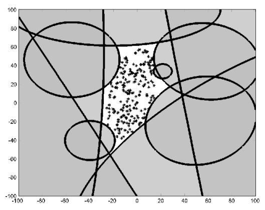
Figure 25: The Evolution of two ellipsoids using self data PD3.

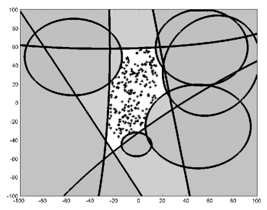

(a) 100 Generations

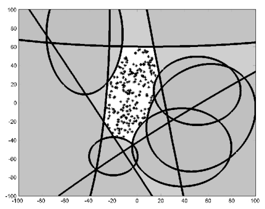

(b) 200 Generations

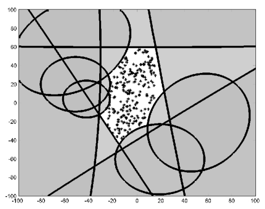

(c) 300 Generations

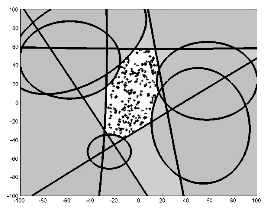

(d) 400 Generations

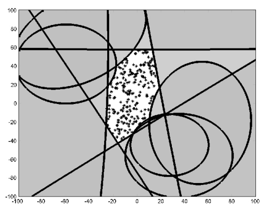

(e) 1000 Generations

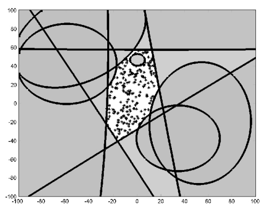

(f) 1200 Generations

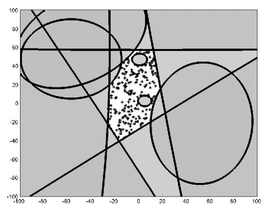

(g) 2000 Generations

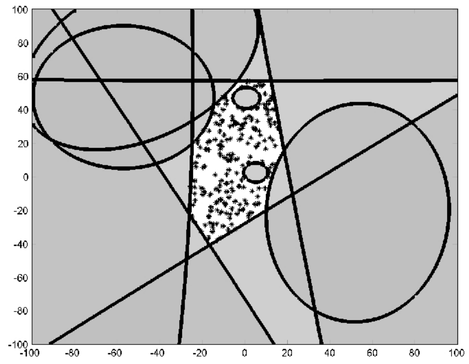

(h) 2500 Generations


(i) 3500 Generations


(j) 4500 Generations

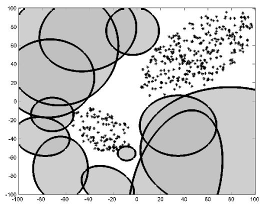

(k) 6000 Generations

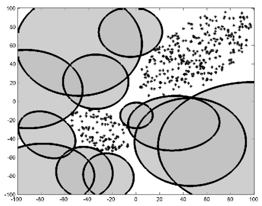

(l) 10000 Generations

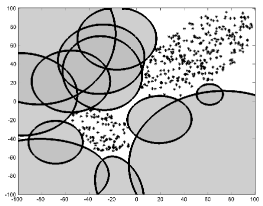


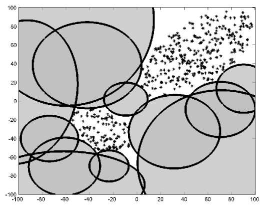
(m) 10000 Generations

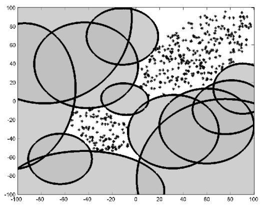
Figure 26: The Evolution of ten ellipsoids using self data set PD4. (m) is an enlarged version of (l). It is enlarged to show the two smaller ellipses inside of the ellipse of data points.

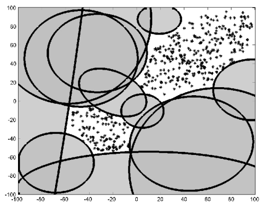

(a) 100 Generations

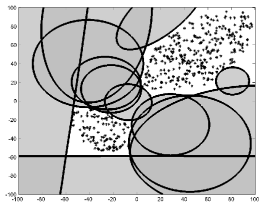

(b) 200 Generations

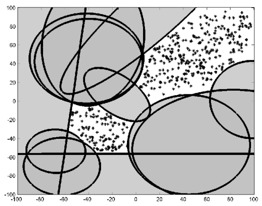

(c) 300 Generations

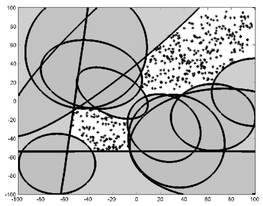

(d) 400 Generations

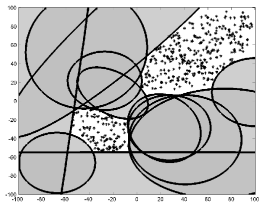

(e) 500 Generations

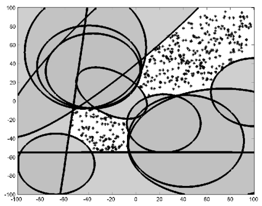

(f) 1000 Generations


(g) 2000 Generations


(h) 4000 Generations


(i) 7000 Generations


(j) 10000 Generations


(k) 15000 Generations

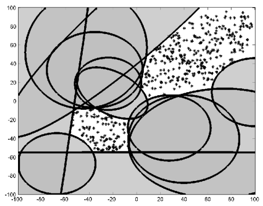
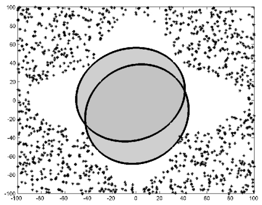
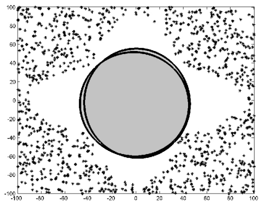
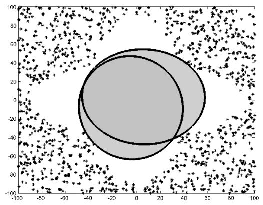
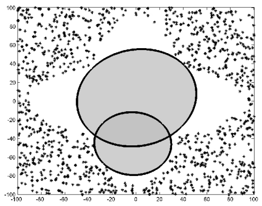

(l) 20000 Generations

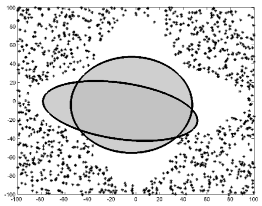
Figure 27: The Evolution of twelve ellipsoids using self data set PD5.

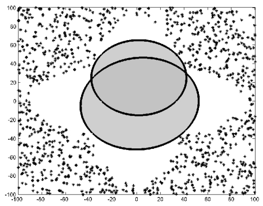

(a) 100 Generations

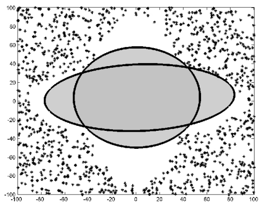

(b) 200 Generations

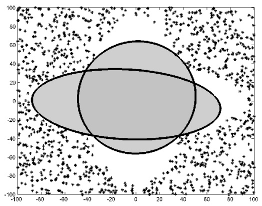

(c) 500 Generations

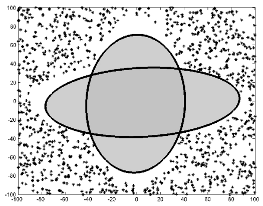

(d) 800 Generations

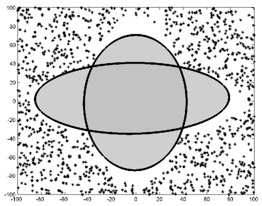

(e) 1100 Generations

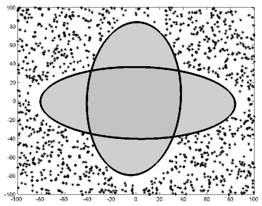

(f) 1500 Generations


(g) 1800 Generations


(h) 2100 Generations


(i) 2400 Generations


(j) 2700 Generations


(k) 5000 Generations

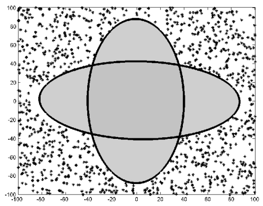
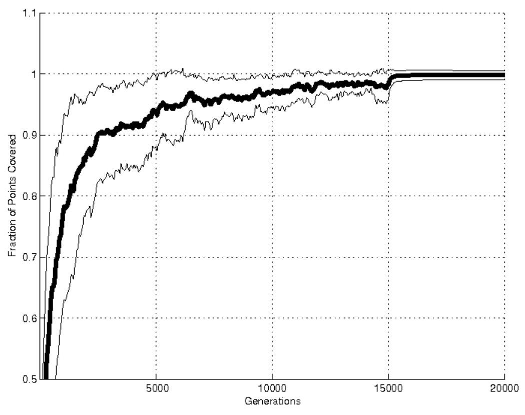
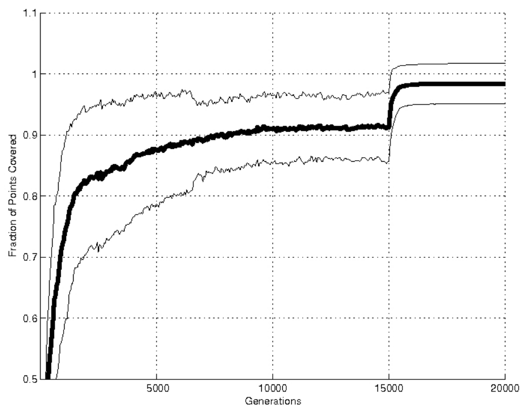

(l) 10000 Generations

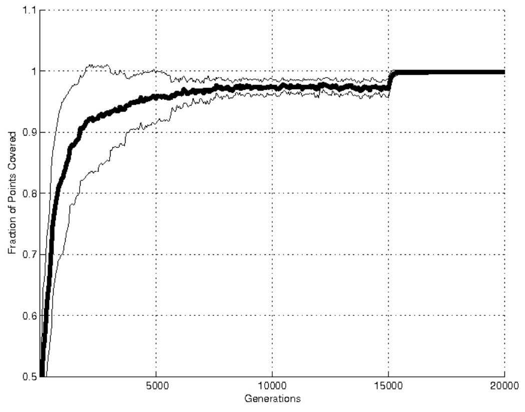
Figure 28: The Evolution of ten ellipsoids using self data set PD6.



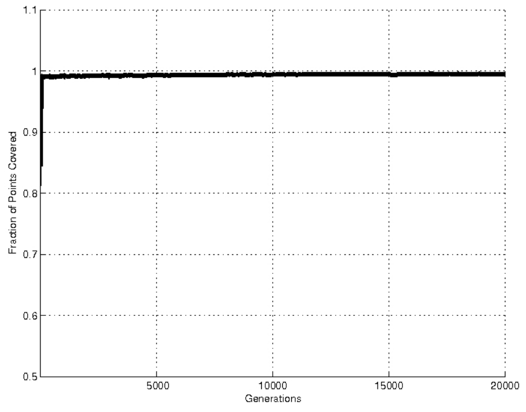
(a) PD1, 1 Ellipsoid



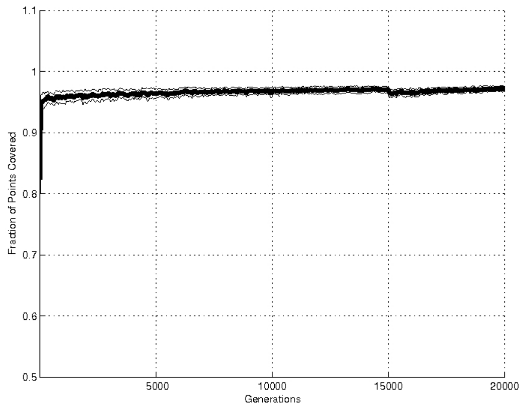
(b) PD2, 2 Ellipsoids



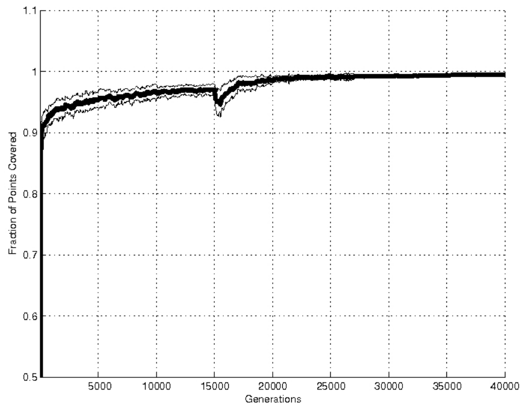
(c) PD3, 2 Ellipsoids



(d) PD4, 12 Ellipsoids

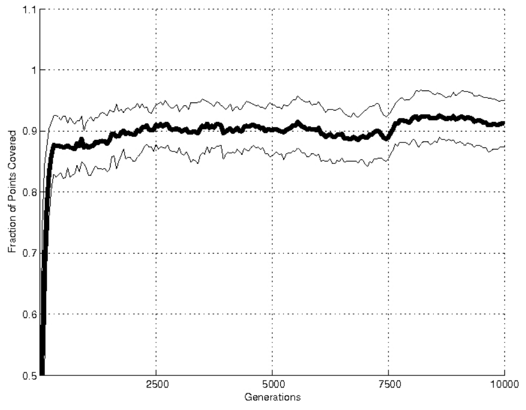


(e) PD5, 20 Ellipsoids

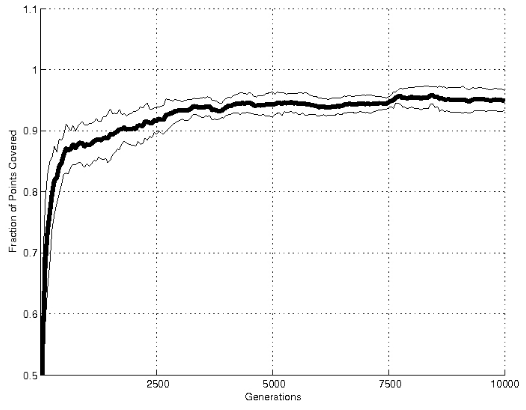


(f) PD6, 2 Ellipsoids

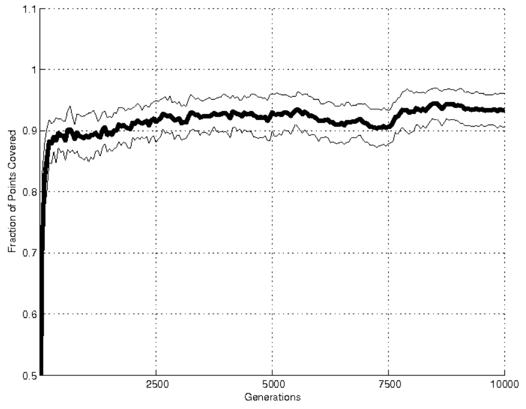
Figure 29: Coverage of Test Points v. Generation, Ellipsoids, Data sets PD1-PD6. The ordinate is the fraction of the nonself test points that the generated ellipsoids cover at the generation specified by the abscissa. The thick line is the average performance over 30 runs. The thin solid lines above and below the thick lines are the average performance plus and minus, respectively, the average of the standard deviation over 30 runs.



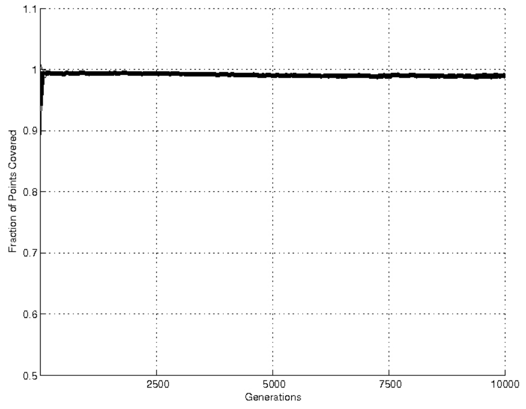
(a) PD1, 6 Spheres



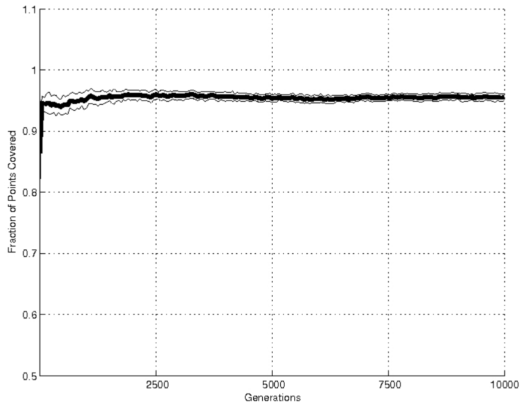
(b) PD2, 16 Spheres



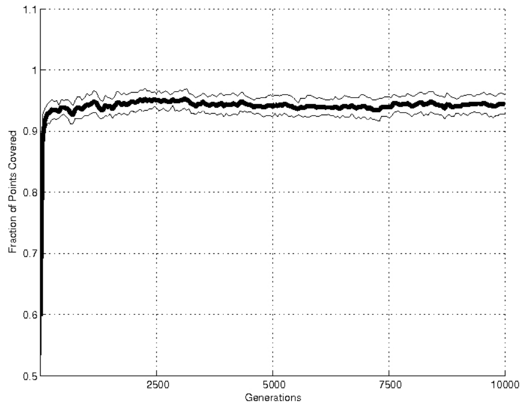
(c) PD3, 14 Spheres



(d) PD4, 20 Spheres

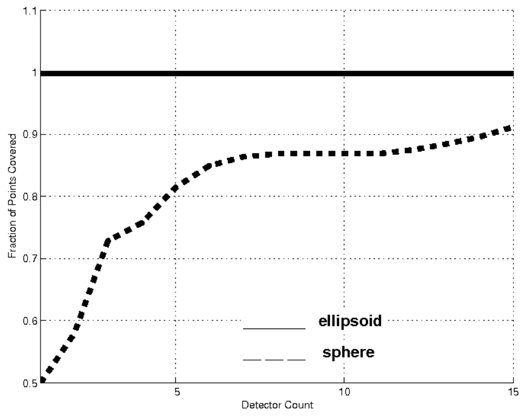


(e) PD5, 20 Spheres

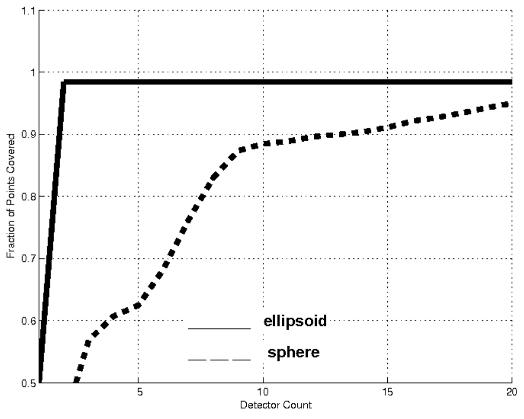


(f) PD6, 18 Spheres

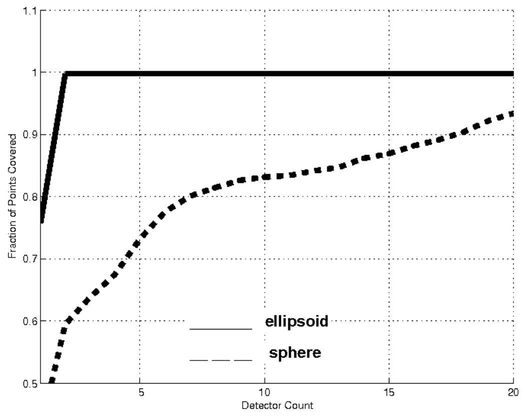
Figure 30: Coverage of Test Points v. Generation, Spheres, Data sets PD1-PD6. The ordinate is the fraction of the nonself test points that the generated spheres cover at the generation specified by the abscissa. The thick line is the average performance over 30 runs. The thin solid lines above and below the thick lines are the average performance plus and minus, respectively, the average of the standard deviation over 30 runs.



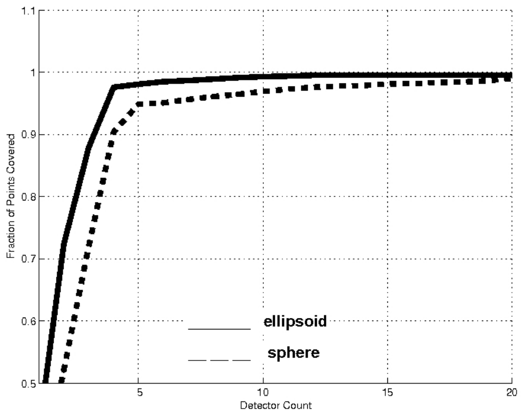
(a) PD1



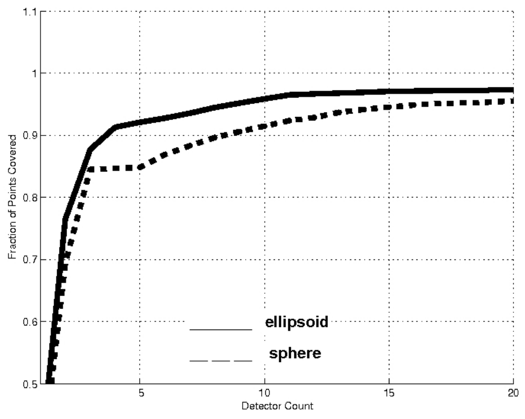
(b) PD2



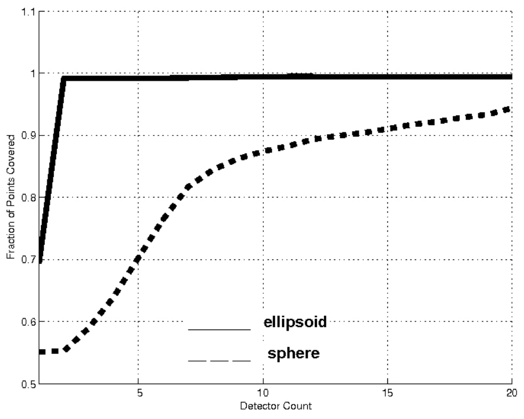
(c) PD3



(d) PD4

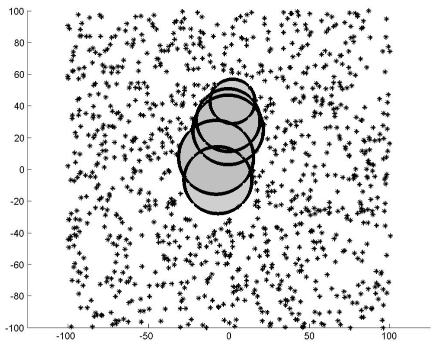


(e) PD5

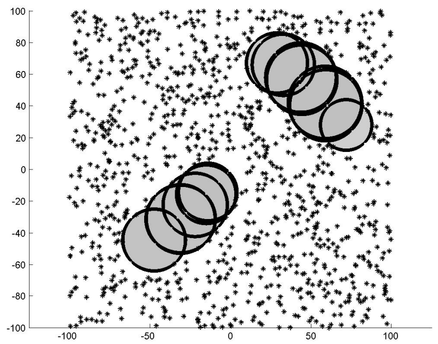


(f) PD6

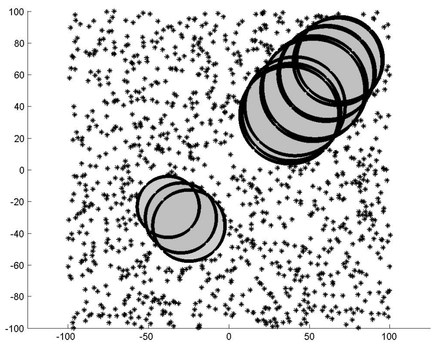
Figure 31: Nonself Coverage v. Detector Count, Ellipsoids and Spheres, Data Sets PD1-PD6.



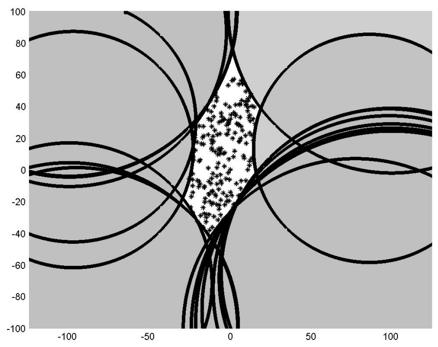
(a) PD1, 15 spheres



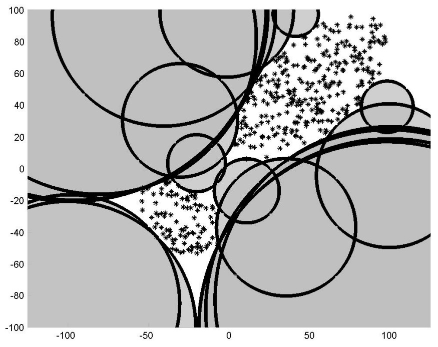
(b) PD2, 20 spheres



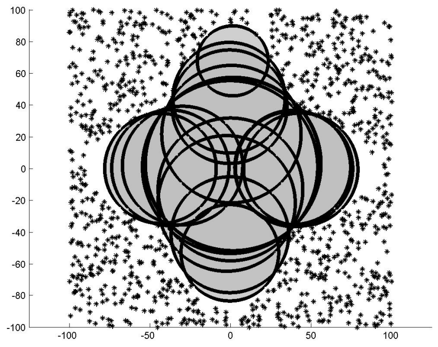
(c) PD3, 20 spheres



(d) PD4, 20 spheres



(e) PD5, 20 spheres



(f) PD6, 20 spheres

Figure 32: Final sphere solution for data sets PD1-PD6.

5.2 MIT Lincoln Labs Data

The Lincoln Labs data shows that this algorithm works on a real world intrusion detection data set. Figures 33 and 35 show that both the ellipsoids and spheres perform successfully against this data set. The false alarm rate for both ellipsoids and spheres is 0. Although ellipsoids are more flexible than spheres, it appears that this data set does not require that flexibility. The number of spheres necessary to achieve good results is about the same as the number of ellipsoids necessary for comparable results. Figure 36 shows this graphically. In fact, Figure 36 shows that there are some numbers of detectors for which spheres perform better than ellipsoids. We hypothesize that this is an artifact of the algorithm and constraints. The constraint that the center point of an ellipsoid must remain inside of the problem domain bounds could cause this to happen. We believe that ellipsoids should perform equal to or better than spheres. This is because ellipsoids are more flexible. However, performance in the EA and performance in a classification testing are slightly different. In the EA, performance (or objective value) is measured by how much nonself space a detector covers. Performance in classification testing is dependent on where the test points are located. That is, an ellipsoid that outperforms a sphere according to the EA criteria (objective value) could be outperformed by the sphere according to the classification testing metric. Figure 34 shows an example of this scenario.

When our algorithm with ellipsoids is run against the self data shown in Figure 34 it generates the ellipsoid in the Figure. However, when our algorithm is run with spheres against the same self data it generates the sphere in the Figure. The best sphere is not in the same location (no overlap) as the best ellipsoid, where "best" in this sentence refers to EA objective value. However, since the nonself test points fall inside of the sphere, it may appear that one sphere performs better than one ellipsoid. This is true only in the classification sense because the ellipsoid actually covers more problem domain space than the sphere.

Figure 37 validates our hypothesis for why spheres perform better than ellipsoids for some detector counts. The figure, along with Figure 36 shows that the ellipsoids always cover more space than the spheres, even though the spheres may classify better. Again, this is simply an artifact of the specific data set.

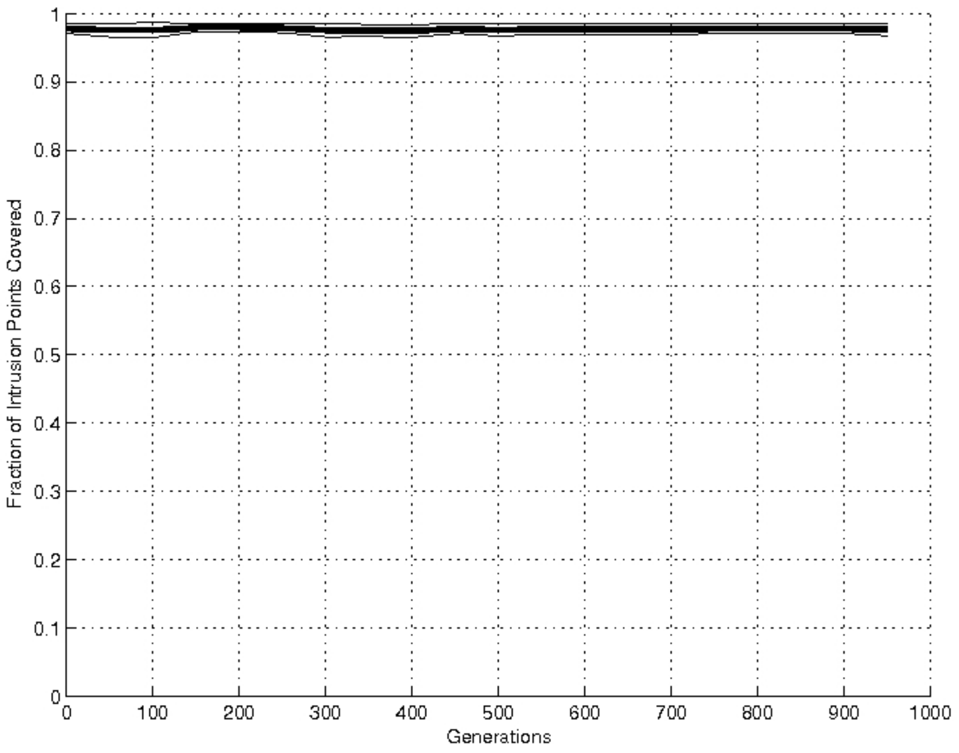


Figure 33: Coverage of nonself intrusion points by Ellipsoids v. Generation, MIT data set. The thick line is the average performance over 30 runs. The thin solid lines above and below the thick lines are the average performance plus and minus, respectively, the average of the standard deviation over 30 runs.

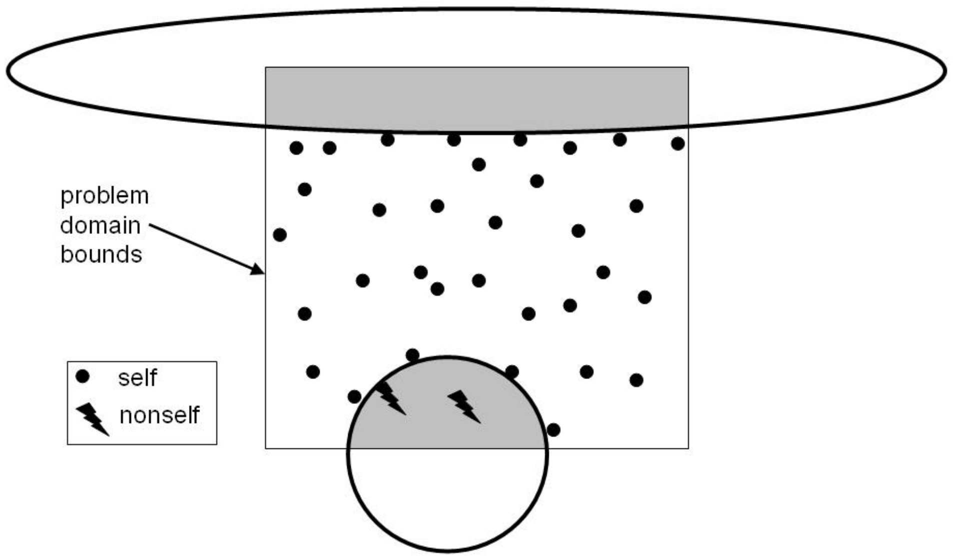


Figure 34: The ellipsoid has a better objective value than the sphere in the EA. However, the sphere performs “better” in classification because the nonself test happen to fall within the area it covers.

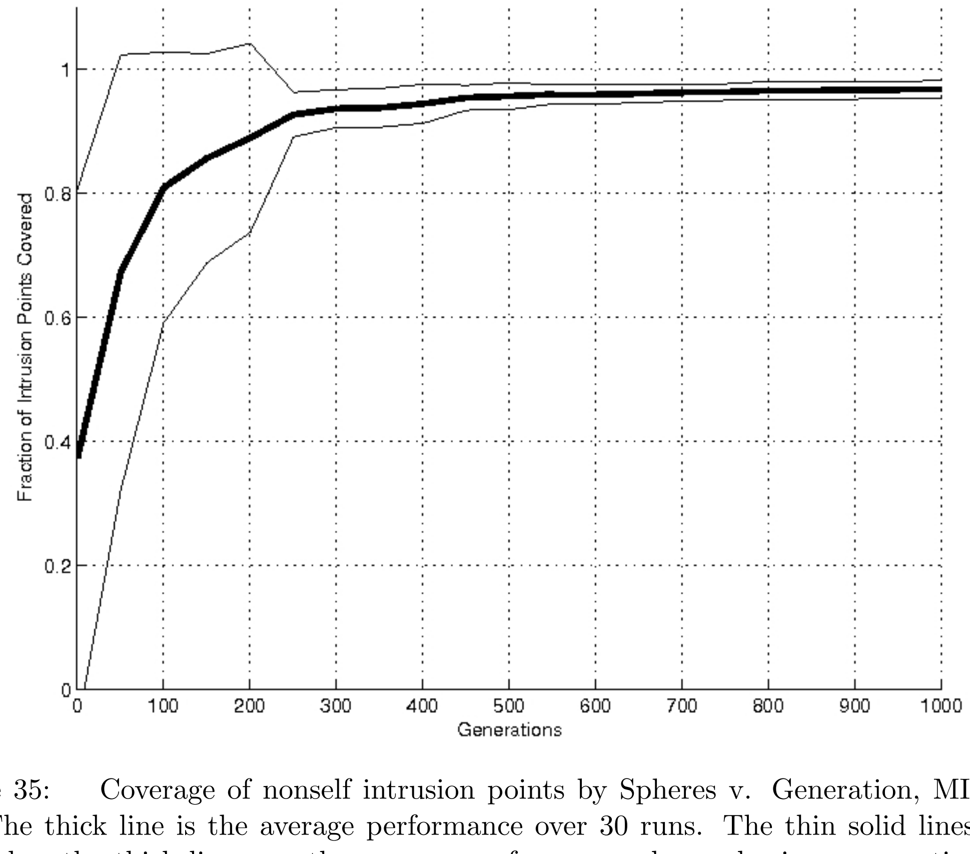


Figure 35: Coverage of nonself intrusion points by Spheres v. Generation, MIT data set. The thick line is the average performance over 30 runs. The thin solid lines above and below the thick lines are the average performance plus and minus, respectively, the average of the standard deviation over 30 runs.

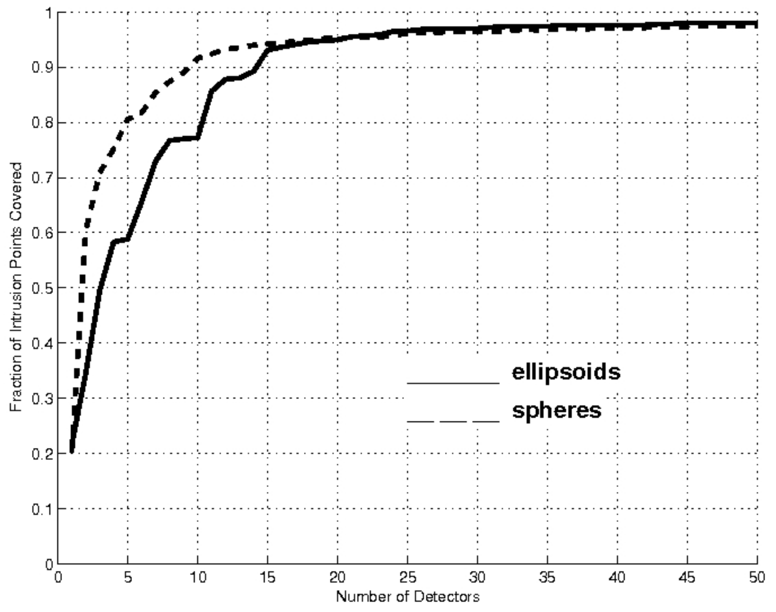


Figure 36: Comparison of the fraction of nonself intrusion points covered by ellipsoids and spheres versus the respective count of each.

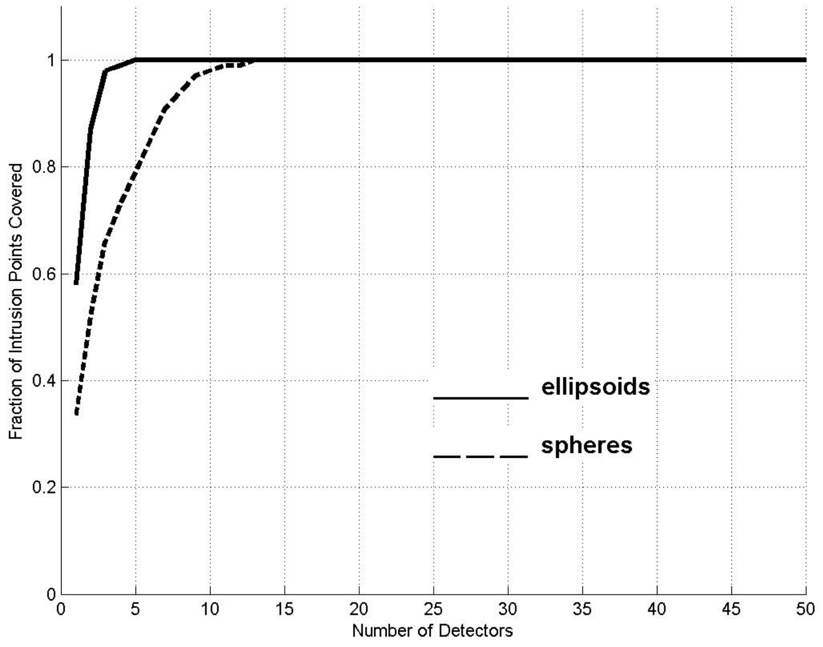


Figure 37: Comparison of the approximate fraction of space covered by ellipsoids and spheres versus the respective count of each.

Table 12: Comparison of several negative selection algorithms on the MIT IDS data set. Our algorithm (ellipsoids, spheres) is compared with EFR (Evolving Fuzzy Rules) [38] and Negative Characterization [41].

Algorithm	**Detection Rate**	**False Alarm Rate**	**# Detectors**	**# Runs**
our Ellipsoids	91.52	0.00	10	30
our Spheres	91.64	0.00	10	30
EFR	98.30	2.0	8.87	10
Negative Characterization	87.5	< 1.0	10	10

We also address the issue of false alarm classifications, i.e. self points being classified as nonself. This subject receives minimal attention because there is no ($< 0.01\%$) false alarm rate. We surmise that this is a result of the different classes being easily separable.

We do not show a receiver operating curve (ROC) for the MIT IDS data because the false alarm rate is so low. This makes the ROC very uninteresting, as it is simply a vertical line (assuming the abscissa is the false alarm rate).

Results from our algorithm also compare well with other negative selection approaches. Dasgupta and Gonzalez [41] report that their negative selection algorithm finds 87.5% of the intrusion points with a false alarm rate of no more than 1%. Using 50 ellipsoids, our algorithm finds 97.45 % of the intrusion points with a false alarm rate of 0.01%. Using 50 spheres, our algorithm finds 97.52% of the intrusion points with a false alarm rate of 0.01%. Dasgupta's results are an average from 10 runs while our results are an average from 30 runs. Hence, our results are statistically different.

5.3 Iris

Figures 38, 39, and 40 show the correct and incorrect classification rates by generation for ellipsoids. As we note in Section 4.1.2, setosa is linearly separable from versicolor and virginica. The results confirm that it is significantly easier to use setosa rather than versicolor or virginica as the self data.

Figures 38, 39, and 40 show, while ellipsoids and spheres converge to the same performance (true positive and false positive), the ellipsoids require more generations to achieve this performance. We hypothesize that this is because this data set may not require the flexibility of ellipsoids. That is, an optimal solution may be achievable with ellipsoids whose semiaxes have similar lengths. This means that the ellipsoid model may waste mutations on orientation mutation and on mutating semiaxis lengths separately, even though these mutations do not lead to an improvement in the solution. Since the sphere model is constrained to disallow orientation mutation and individual semiaxis length mutation, spheres are able to more quickly complete the radius and center point mutations that are necessary to obtain a good solution.

Figures 41, 42 and 43 show nonself detection rates and false alarm rates for spheres and ellipsoids when setosa, versicolor and virginica are used, respectively, as the self class. When setosa is used for training, the problem is so easy that the flexibility of the ellipsoids do not result in any advantage. Ellipsoids and spheres both achieve almost perfect performance only one detector. The greatest advantage for ellipsoids occurs when versicolor is used for training. The ellipsoids classify about 5% more nonself points correctly, although the ellipsoid algorithm also has a slightly higher false alarm rate. Figure 43 shows that, when virginica is used for training, the spheres and ellipsoids achieve similar performance in nonself detection rate while spheres perform better in false alarm rate. We hypothesize that the flexibility of the ellipsoids may result in overtraining on this data set.

Figures 44, 45 and 46 show the receiver operating curves (ROC) for the different training classes. These ROC curves are simply a plot of the false alarm rate v. correct nonself classification rate. The number of points in each curve is simply the number of ellipsoids that produce that result. If the algorithm runs with n ellipsoids, then there is a point for the best ellipsoid, two best ellipsoids, three best ellipsoids,..., n best ellipsoids.

These curves validate that the ellipsoid algorithm can successfully differentiate between classes in the iris data set. These ROCs show that our results for ellipsoids and spheres are relatively similar. The only training class for which there is a difference in the ROCs is versicolor. When training on versicolor, the ellipsoid algorithm performs better than the sphere algorithm by the ROC metric.

Only one research effort [49] which uses a negative selection algorithm with the iris data set is found for comparison. In this research, Dasgupta and Ji use negative selection

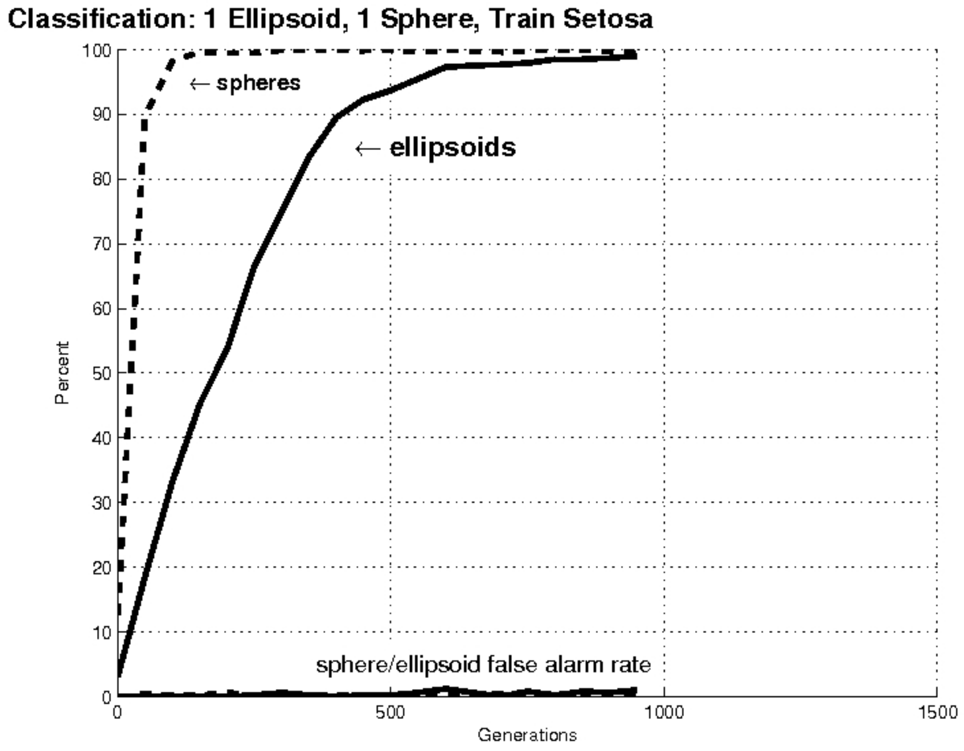


Figure 38: The upper two lines represent the percent of nonself (versicolor and virginica) test points classified correctly using ellipsoids and spheres. The lower two lines represent the percent of self (setosa) test points classified incorrectly using ellipsoids and spheres.

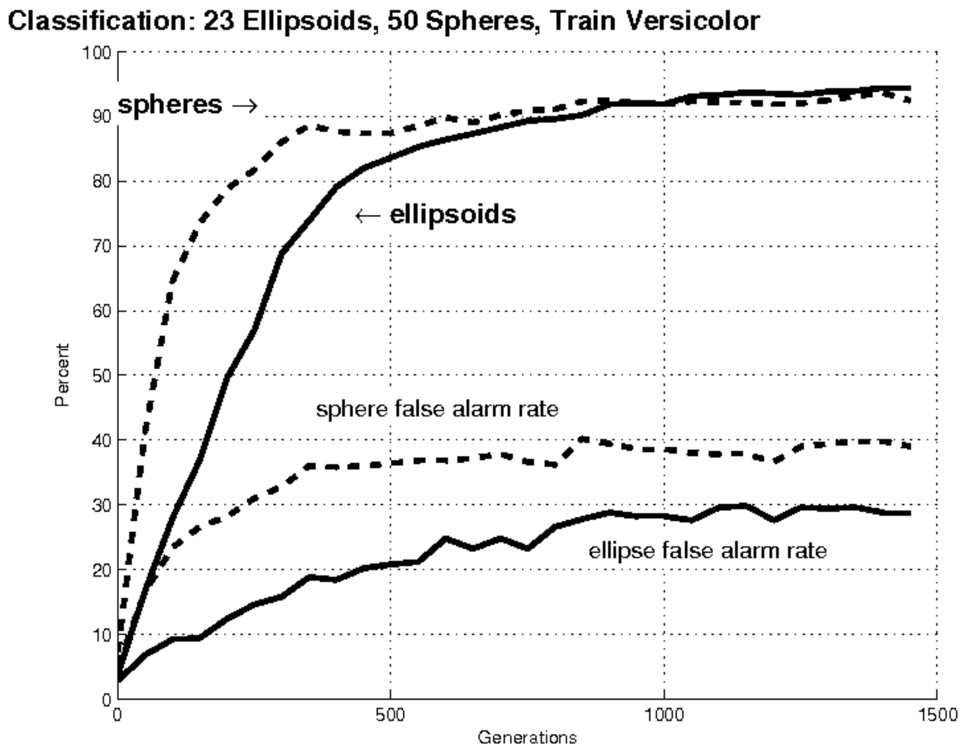


Figure 39: The upper two lines represent the percent of nonself (setosa and virginica) test points classified correctly using ellipsoids and spheres. The lower two lines represent the percent of self (versicolor) test points classified incorrectly using ellipsoids and spheres.

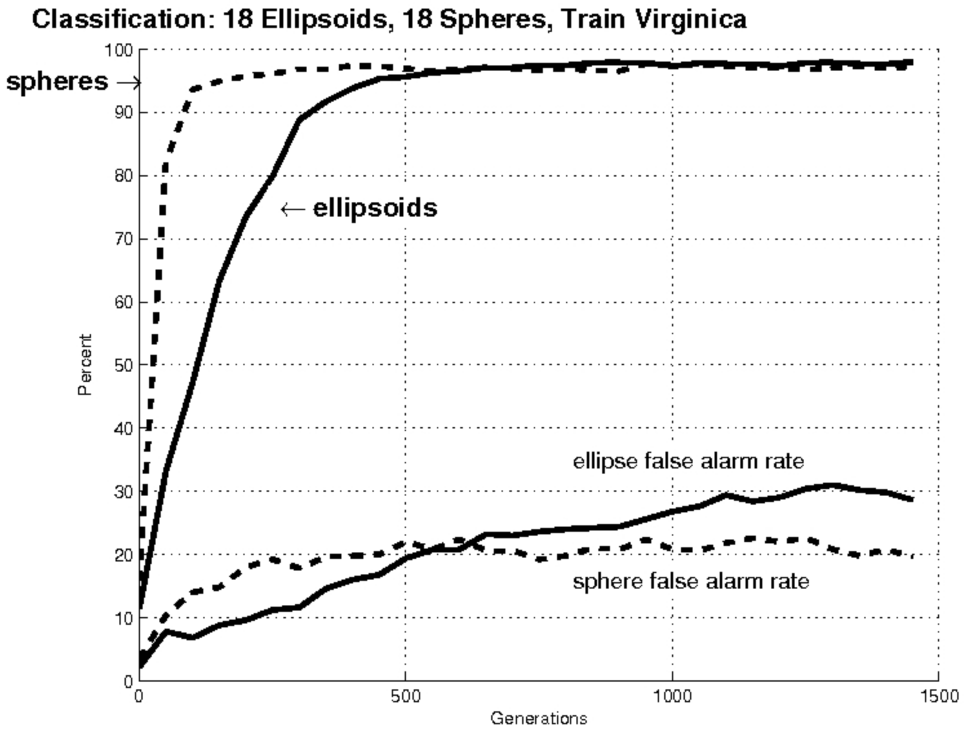


Figure 40: The upper two lines represent the percent of nonself (setosa and versicolor) test points classified correctly using ellipsoids and spheres. The lower two lines represent the percent of self (virginica) test points classified incorrectly using ellipsoids and spheres.

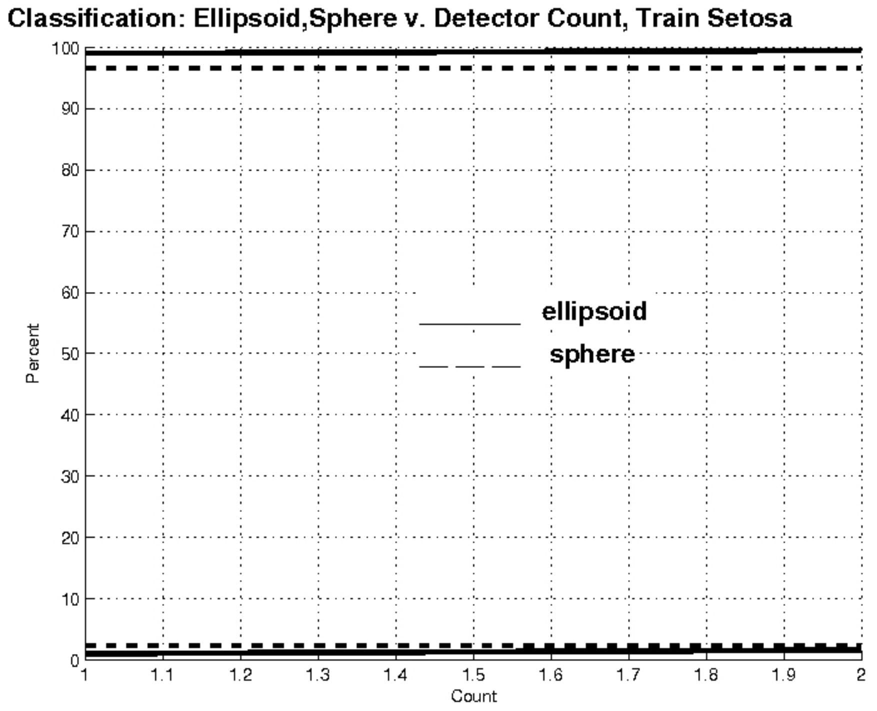


Figure 41: The upper two lines represent the percent of nonself (versicolor and virginica) test points classified correctly using ellipsoids and spheres. The lower two lines represent the percent of self (setosa) test points classified incorrectly using ellipsoids and spheres. The abscissa is the number of detectors that achieve the results shown.

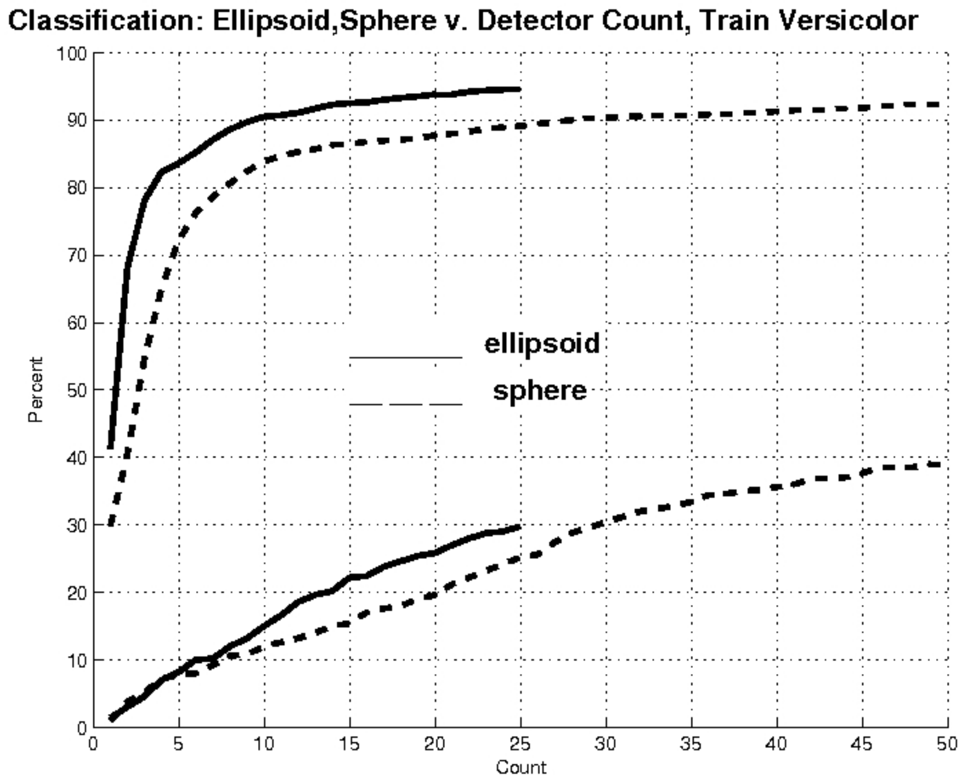


Figure 42: The upper two lines represent the percent of nonself (setosa and virginica) test points classified correctly using ellipsoids and spheres. The lower two lines represent the percent of self (versicolor) test points classified incorrectly using ellipsoids and spheres. The abscissa is the number of detectors that achieve the results shown.

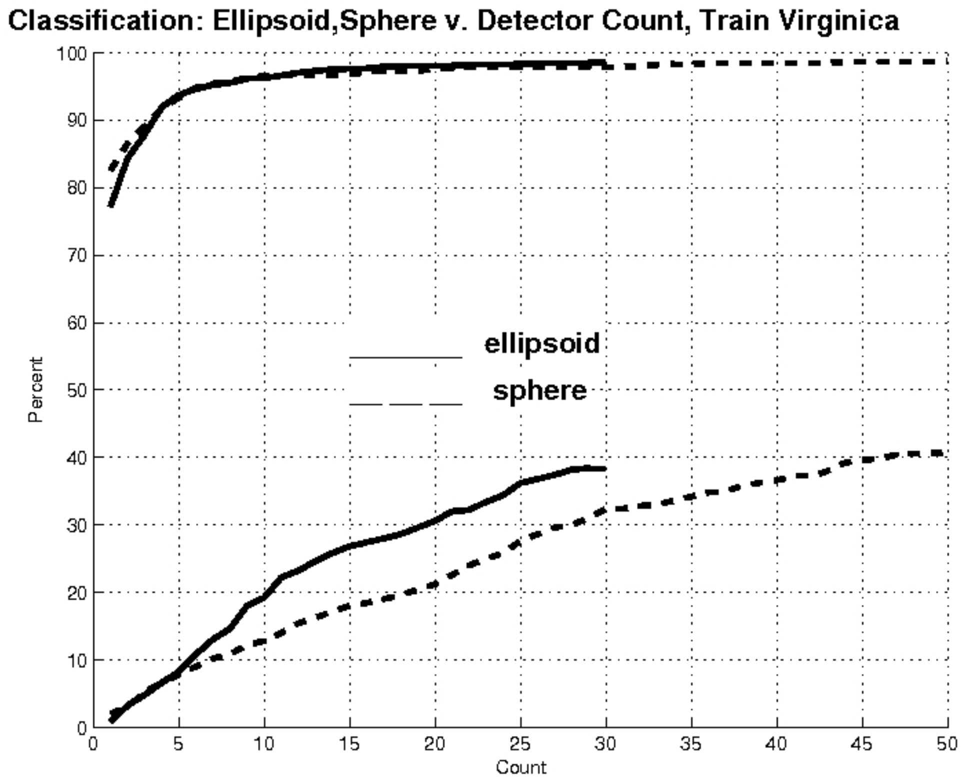


Figure 43: The upper two lines represent the percent of nonself (setosa and versicolor) test points classified correctly using ellipsoids and spheres. The lower two lines represent the percent of self (virginica) test points classified incorrectly using ellipsoids and spheres. The abscissa is the number of detectors that achieve the results shown.

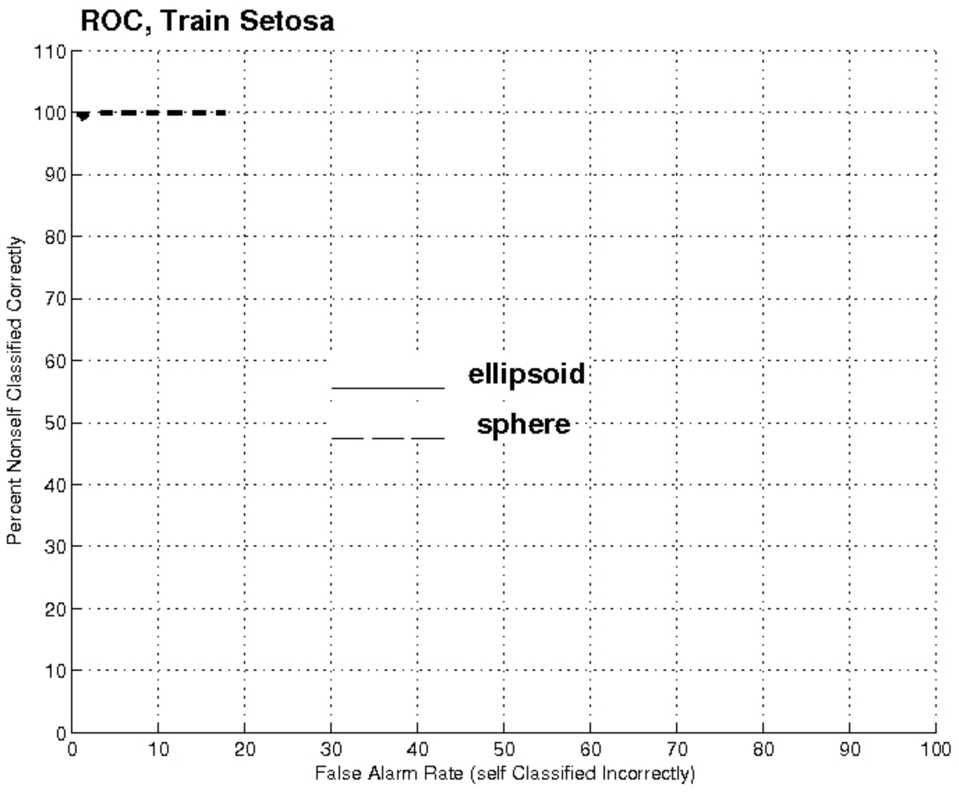


Figure 44: The ROC curve for ellipsoids and spheres when training on setosa (self). Nonself is versicolor and virginica. The abscissa is the false alarm rate (percent of self points incorrectly classified. The ordinate is the percent of nonself points correctly classified.

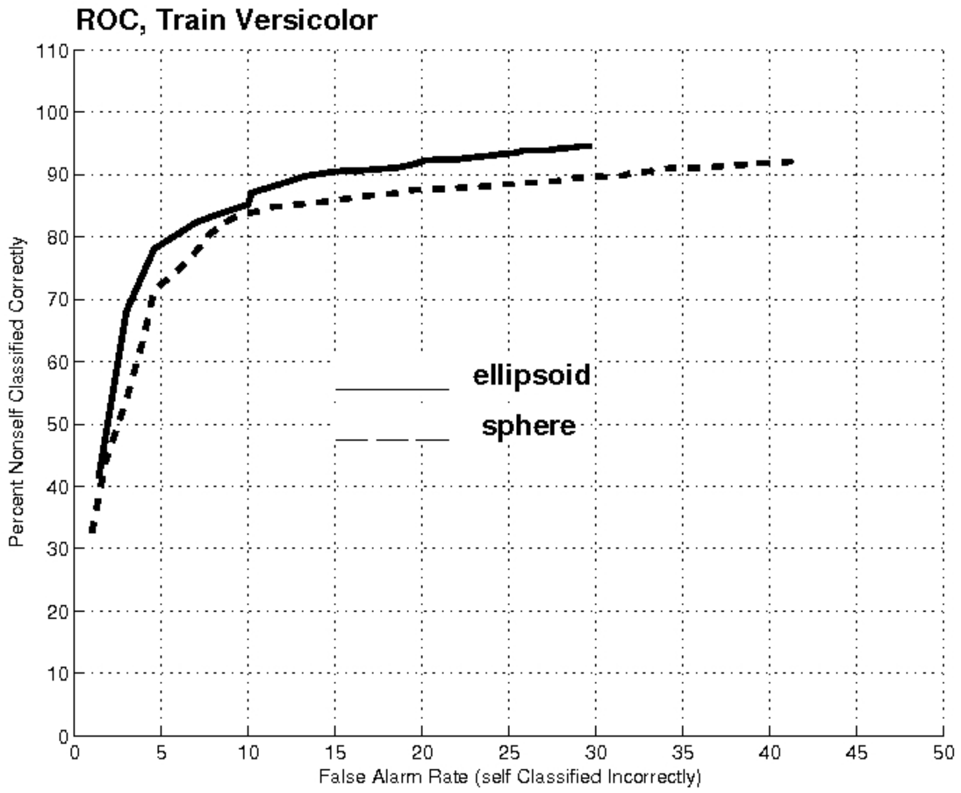


Figure 45: The ROC curve for ellipsoids and spheres when training on versicolor (self). Nonself is setosa and virginica. The abscissa is the false alarm rate (percent of self points incorrectly classified. The ordinate is the percent of nonself points correctly classified.

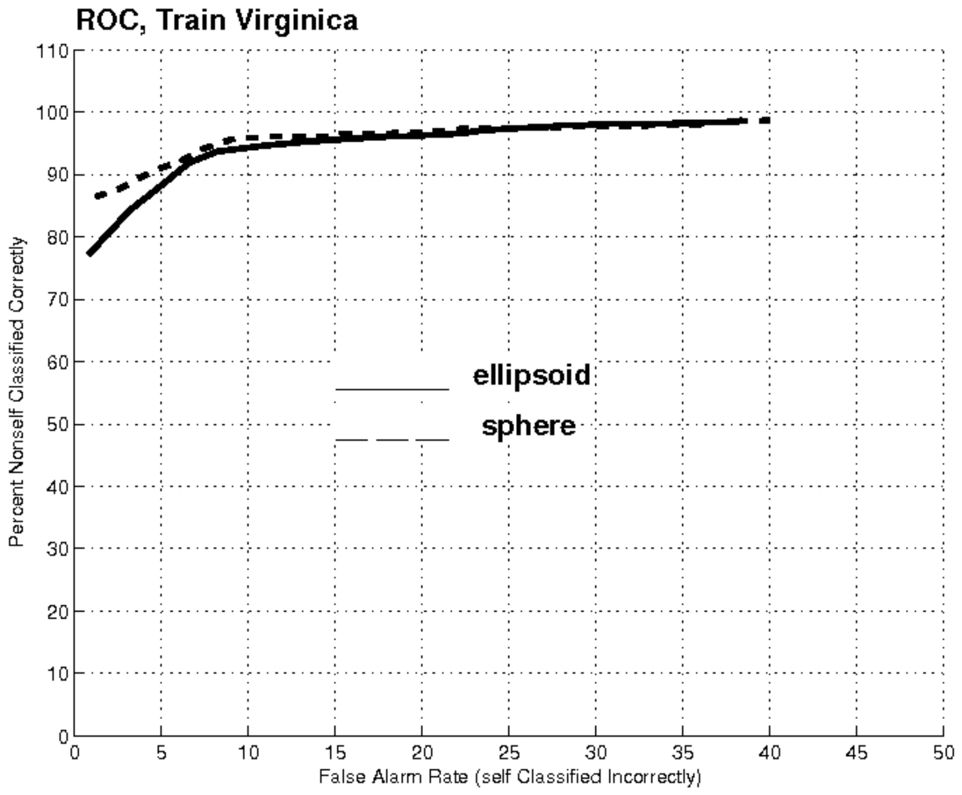


Figure 46: The ROC curve for ellipsoids and spheres when training on virginica (self). Nonself is setosa and versicolor. The abscissa is the false alarm rate (percent of self points incorrectly classified. The ordinate is the percent of nonself points correctly classified.

with variable sized detectors (hyper-spheres) as the detectors. A summary of their results compared with our results is shown in Table 13. Comparison with V-detector results reflects the ellipsoid algorithm's ability to find a set of detectors that efficiently covers nonself space.

Table 13: Comparison of spherical and elliptical detectors. V-detector is Dasgupta and Ji's [49] algorithm.

Training Data	**Alorithm**	**Detection Rate**		**False Alarm Rate**		**Detectors**	**Runs**
		Mean	**SD**	**Mean**	**SD**		
Setosa	**Spheres**	99.80	1.41	0.20	2.00	1	100
	Ellipses	99.20	3.39	0.60	3.43	**1**	100
	V-detector	99.97	0.14	1.32	0.95	20	100
Versicolor	**Spheres**	90.90	10.36	32.00	21.84	31	100
	Ellipses	90.80	10.20	17.00	16.67	**10**	100
	V-detector	88.30	2.77	8.42	2.12	110.08	100
Virginica	**Spheres**	98.00	4.02	33.80	21.21	36	100
	Ellipses	98.00	4.26	30.00	18.09	**20**	100
	V-detector	93.58	2.33	13.18	3.24	108.12	100

VI. Conclusion

This chapter presents a summary of the research discussed in this document. Significant contributions of this research are outlined and recommendations for future research are provided.

6.1 Research Problem

Network security is becoming more and more important because of the ubiquity of networks and the value of the information stored on networks. For these reasons, protection against network intrusions is very important. Currently, most networks use signature detection software. However, signature detection only protects against previously encountered intrusions for which a signature update has been issued.

This type of protection is insufficient because many intrusions are new or variations of older intrusions. Either way, a signature detection system does not recognize such as intrusions and damage may be incurred. Anomaly detection is an alternative to signature detection that has the potential to protect against new intrusions.

In an anomaly detection problem, a detection model is built from innocuous network data. Then, this model is used to monitor new data. If a datum is statistically different from the model, it is labelled as an anomaly and action is taken. This problem may be stated as a classification in which only one class (self) is available for training. The model that results from the training phase is then used to detect anomalies during the testing stage.

Earlier research noticed a similarity between this problem and the way that the body protects itself from harmful invaders. This observation led to artificial immune system (AIS), a computational paradigm that uses biological immune system (BIS) inspired systems to find solutions to various computational problems. This research focuses on the negative selection algorithm, which generates nonself detections during the training phase.

Previous work has validated the potential of the negative selection algorithm. This work looks at geometric detectors. Previous geometric detectors include hyper-rectangles and hyper-spheres. This research extends to ellipsoids (hyper-ellipses).

The following objectives are outlined for this research:

- Establish a mathematical model to define and manipulate ellipsoids.
- Develop a negative selection algorithm to generate a set of ellipsoid detectors by training on self data.
- validate our model qualitatively by testing on 2d data sets and quantitatively by testing on higher dimension real world data sets. Also, validate that our algorithm works on network intrusion data.

This research successfully accomplishes the above objectives:

- Ellipsoid Model: Section 3.2 establishes a model that can be used to describe any n-dimensional ellipsoid. We also define operations to define find the volume of an ellipsoid and to establish whether or not a point is in an ellipsoid.
- Develop a negative selection algorithm: Generating detectors is determined to be a complex problem with no known deterministic algorithm that runs in tractable time. For this reason, an EA is designed to generation ellipsoid detectors. The EA uses several important data structures, including a 2^n-way tree for ellipsoid approximations. Approximations are important since computing ellipsoid overlap analytically is very complex. The algorithm is described in detail (see Chapter III) and pseudocode is provided in Appendix C.
- Validation: Results in Chapter V show that our algorithm finds good ellipsoids that cover nonself space. This is shown graphically on pedagogical data sets PD1-PD6. Also, numerical result analysis from all data sets shows that our algorithm can find good ellipsoid detectors. Further, results from network intrusion data show that our negative selection has potential to work in the network intrusion detection domain

6.2 Contributions

This research makes several important contributions:

6.2.1 Pedagogical Data Sets. We describe a system that can be used for classifying negative selection algorithm test data sets. This is important because current research efforts do not test on standard data sets. This makes comparison between different negative selection algorithms difficult. With our test data set taxonomy, researchers can choose different data sets to test specific characteristics of their algorithm. We design six data sets with varying characteristics (PD1-PD6) and use these for testing.

6.2.2 Algorithm to Evolve Ellipsoids. No other algorithm to evolve ellipsoids exists in literature. Our algorithm defines and analyzes mutation operators, an objective function, and convergence metrics.

6.2.3 2^n-way Tree for Ellipsoid Approximations. Computing the amount of overlap between a set of ellipsoid is a difficult that requires an approximation if it is to be accomplished in reasonable time. We implement a novel approach, using a data structure that was introduced in computational geometry. Our approximation achieves good results and allows flexibility in the tradeoff between results and algorithm complexity.

6.2.4 Ellipsoids as Detectors. Using ellipsoids as detectors is important because they are very flexible and allow for more efficiency in a negative selection based detection system. Our results show that ellipsoids are as good spheres and often better, depending on the data set used for testing.

6.3 Future Work

This section presents a list of suggestions for future research.

- Improvements to the ellipsoid algorithm: For instance, the algorithm that finds the closest self point to a randomly chosen center point for a new hyper-ellipsoid currently searches through all of the self points. There is a way to figure out which "segments" in the segment tree are closest so that they can be checked first. Doing this will greatly speed up the generation of children.

- Investigate the possible use of crossover in the evolutionary algorithm to evolve a set of ellipsoids.

- Theory behind parameter choices. e.g., How long to wait before lowering the mutation standard deviation? This depends on the number of ellipsoids in the population and the number of dimensions and probably some other factors. This is because we lower the mutation standard deviation whenever a certain number of generations passes without any individual in the population experiencing improvement. When there are more individuals in the population, it is less likely that a generation will pass without at least one individual experiencing improvement. Therefore, as the size of the population increases, we should decrease the number of generations between checks to see if we should lower the mutation standard deviation. Also, what about the dimensionality of the problem? As the dimensionality increases, is it easier or harder to find a good mutation? It seems that as dimensionality increases it may be easier to find a good mutation because there are more possible valid mutations. However, this is just a guess based on observation and intuition.

- Depending on the density of the self data, there may be "holes" in self space such that a detector that covers that hole receives a larger objective value than a lower objective value detector that is contributes to the cover of nonself space. Future work could investigate techniques for alleviating this issue.

- Testing on other data sets-other geometric shapes (rectangles, stars?, etc.) and variation in data densities could be used. In short, vary the parameters listed in Table 10 in Section 4.2.

- What hyper dimensional shapes, other than rectangles, spheres and ellipsoids can be used to model nonself space in a negative selection problem?

- Can different hyper dimensional shapes be used together (e.g. rectangles and ellipsoids)?

- Analysis and improvement on the approximations in our algorithm. The most significant approximation is the use of the segment tree to approximate area and overlap of ellipsoids. Is there another approach that could be used? What improvements could be made to the segment tree? What about a random shift in the segment tree every generation so that the error is averaged over different ellipsoids. What about a non-uniform segment tree? It is really only necessary to maintain a deep segment tree for areas in the n-space in which there are many details that need to be modelled (e.g. ellipsoid edges). Would it be possible to maintain the segment tree at minimum depth except for areas in n-space where more granularity is needed?

- Use gene libraries as in the BIS with the crossover operator [35]. The building blocks in the gene libraries could be manipulated using a genetic algorithm that allows for explicit manipulation of building blocks, like Goldberg's messy GA [37]
- Development and testing of deterministic algorithms for ellipsoid detector generation. Deterministic algorithms should be compared against stochastic algorithms.
- Development of a technique to dynamically choose the optimal number of ellipsoid detectors.

6.4 Summary

Ellipsoids are a useful geometric structure for nonself detectors because of their flexibility. However, good approximation and generation algorithms are needed to make use of their potential. We show that such algorithms and data structures exist, as well as paving the way so that future research may make ellipsoid use feasible in a real-world application.

There are generally two schools of thought in immune-system network theory: (1)clonal selection model and (2)network immune model. The clonal selection model is the more commonly accepted and used in artificial immune systems. The clonal selection model models interactions between antigens and antibodies [92]. The basic premise of the clonal selection model is that the immune system is relatively inactive until it is stimulated for some foreign invader, i.e. an antigen. The clonal selection model is a somewhat younger and more controversial model. The fundamental principal of the network immune model is that, even when no antigens are present, the immune system is constantly active. The lymphocytes can recognize each other. Interactions between lymphocytes excite and inhibit, depending on whether or not a lymphocyte recognizes or is recognized. The network immune model is often represented by a set of differential equations, each equation describing the change in concentration of each type of lymphocyte (identical receptor) relative to time. A general model of an immune network is given in Equation 46 [20].

$$\begin{array}{c}\text{Rate of}\\\text{population}\\\text{variation}\end{array} = \begin{array}{c}\text{Network}\\\text{stimulation}\end{array} - \begin{array}{c}\text{Network}\\\text{suppression}\end{array} + \begin{array}{c}\text{Influx}\\\text{of new}\\\text{elements}\end{array} - \begin{array}{c}\text{Death of}\\\text{unstimulated}\\\text{elements}\end{array} \tag{46}$$

Danger theory is a third school of thought that is currently gaining support [12] [11]. Dr. Polly Matzinger [65] is currently the principal proponent of danger theory in the BIS. Danger theory is different than traditional immunological theory because it does not adhere to the idea of self/nonself discrimination. That is, danger theory promotes the idea that the BIS does not react to every foreign entity. If this is the case, then questions like "why does the body not react to new foods that we eat? " must be answered. Danger theory proposes that the immune system reacts to "danger signals" and then works to neutralize offending entities in the locality of the danger signal. Although this model does answer some difficult questions raised by the traditional self/nonself model, it is unclear what exactly the danger signals are and where/by whom they are produced. This theory is young and controversial in the immunology community.

A.1 Overview

The human biological immune system has several characteristics that make it a good model for computation: [20, p.55-6] enumerates 19 characteristics of the BIS that make it attractive for computational reasons. In the interest of simplicity, these characteristics are abstracted into several general qualities for discussion. These qualities are distributed nature, memory, multi-layered architecture, preventative, fuzzy pattern recognition. Through mutations and other variations (are there others?), the BIS can produce antibodies to match any antigen. Even with this potential, the number of antigens that can be matched by the existing lymphocytes in a BIS at a given time is very small. Part of the wonder of the BIS is that it can successfully accomplish the task of defending against an almost infinite number of antigen possibilities while maintaining such a small number of lymphocytes.

A.1.1 Distributed with no Central Control. The human biological immune system (BIS) has no central control. Instead, it is distributed throughout the body, but somehow it works together in a unified manner and it is very effective at what it does: protecting the body from harmful intruders.

A.1.2 Memory. When a B-cell binds to an antigen it is not yet "stimulated" in the full sense of the word. The B-cell is "stimulated" when it has bound to an antigen and it receives a co-stimulatory signal from an accessory cell, such as a T-helper cell. Upon, stimulation, the B-cell begins to divide into two types of cells: Terminal (non-dividing cells) cells called plasma cells and B-memory cells. Plasma cells do not divide but they secrete antibodies at a much faster rate than the dividing B-cells, which are also secreting antibodies. B-memory cells circulate through the body and do not secrete antibodies. They are called memory cells because they do not need to recognize an antigen before they are stimulated by a second antigenic stimulus (co-stimulatory). When stimulated by a second antigenic stimulus, B-memory cells rapidly differentiate into plasma cells capable of producing high affinity antibodies.

One other aspect of memory in the BIS is its associativity: Even though B-cells have receptors that only match one specific antigen epitope pattern, they can present a more efficient response to any structurally related antigen. This means that the body maintains some protection against new antigens that are related (maybe slightly mutated versions) of previously encountered antigens. This phenomenon of providing a more efficient secondary response to an antigen structurally rlated to a previously sen antigen is called immunological cross-reaction, or cross-reactive response. This is also related to the pattern matching capabilities of the BIS because of the principal of generality. That is, the BIS's response to antigens is generalized so that it can react quickly when it encounters an antigen that is related to a previously encountered antigen.

A.1.3 Multi-Layered. The BIS can usually be split up into three layers: the physical layer, the innate layer, and the adaptive layer.

A.1.3.1 Physical Layer. The physical layer is made up of the skin and the respiratory system. The skin performs obvious functions is keeping intruders outside of the body. The respiratory system assists by trapping irritants in nasal hairs, carrying mucus out of the body, and coughing and sneezing.

A.1.3.2 Innate Layer. The innate layer is the next layer under the physical layer. It is non-adaptive, meaning that it does not change relative to current or previous intruders of infections. This is different from the adaptive layer, which is discussed in the next section. The innate immune system plays an important role in controlling infections during the period between an infection's introduction to the body and the time (usually a couple of days later) when the infection is attacked by components of the adaptive layer. Phagocytic cells (e.g. macrophages and neutrophils) have surface receptors that recognize and bind to common molecular patterns found only in foreign microorganisms. After a phagocytic cell has bound to an intruder, lymphokines are secreted. Lymphokines are a type of cytokines, proteins emitted by cells that affect the behavior of other cells. Secreted lymphokines induce an inflammatory response, which draws immune cells, such as phagocytosis, to the infection. The intruder is then attacked, often by phagocytosis, which digests foreign organisms.

A.1.3.3 Adaptive Layer. The lymphocyte is the most important cell in the adaptive layer of the BIS. Lymphocytes are produced in the bone marrow and thymus and they carry specific antigen receptors. The development of lymphocytes uses a mechanism that rearranges its gene receptors so that there is a large number of possible receptors that each lymphocyte can have. B-cells are produced in the bone marrow. The generation of B-cells uses gene libraries to pull genes from for new B-cells. When a lymphocyte binds with an antigen, it is activated and produces a large number of offspring, each of which produce antibodies with the same antigen receptors. This is why lymphocytes are part of an adaptive layer: The number of lymphocytes that match a specific antigen grows as matching lymphocytes bind with the antigen. Lymphocytes that bind with antigens are usually kept around as memory cells, the idea being that if an antigen was seen once, there is a higher likelihood that it will be seen again. By keeping matching lymphocytes around, the body's response will be much faster the next time the antigen is encountered.

The clonal selection principal describes how the BIS adaptively responds to an antigenic stimulus. When a lymphocyte is activated by binding to an antigen, it needs to produce additional lymphocytes with the same receptor specific cells to fight against the infection. This production of additional lymphocytes to fight infection, called clonal expansion, takes place in the lymph nodes. Through clonal expansion, only those lymphocytes that recognize an antigen will proliferate and reproduce. In this way, a kind of selection takes place because the fittest cells, i.e. those that recognize antigens, are selected against the cells that do not recognize antigens.

Clonal selection operates on both B-cells and T-cells, however there is a significant difference. B-cells undergo somatic mutation during reproduction. Mutation helps B-cells to increase their diversity so that they can better fight against selective antigens. T-cells do not suffer mutation during reproduction, so the receptors on child cells are identical to the receptors on parent cells.

A.1.4 Preventative. The BIS can recognize antigens that it has never even seen before. Although clonal selection is adaptive in that it helps the body to have greater numbers of the lymphocytes that it needs to battle current infections, it does not adapt to possible future attacks. This prevention is accomplished through a process called affinity maturation. During clonal expansion, the receptors on B-cells are mutated in three ways: point mutations, short deletions, sequence exchanges. The mutated B-cells are checked for affinity, a measure of how general a B-cell is. That is, how many antigens will elicit a secondary response from this B-cell's receptor. If a mutated B-cell has high affinity, then it enters the pool of memory cells (B-memory cells). If a mutated B-cell has low affinity or is self-reactive, it must be destroyed because it is not useful and could attack self cells. It is believed that these poor-affinity B-cells die through apoptosis (cell death) in the germinal centers (the places where the B-cells are undergoing affinity maturation).

A.1.5 Pattern Recognition. The body has two kinds of lymphocytes, B-cells and T-cells. B-cells and T-cells carry surface receptor molecules that can recognize antigens. These receptors recognize a portion of an antigen called an epitope. Antigens have several different epitopes, so they can be bound by several different lymphocyte receptors. B-cells have receptors that recognize antigens when they are free in solution. T-cells, however, recognize antigens when they have been processed and bound to a molecule called major histocompatibility complex (MHC). There are two classes of MHCs, MHC-I and MHC-II. When B-cells recognize an antigen, they secrete antibodies with receptors that recognize the same antigen pattern. The binding of an antibody to an antigen is a signal to other cells to ingest, process, and/or remove the bound substance. Killer T-cells recognize antigens bound to MHC-I molecules and helper T-cells recognize antigens bound to MHC-II molecules.

The B-cell receptors go through a rearrangement process and mutation so that there is a large number of different receptors that can be created from a finite genome.

A.1.6 Positive Selection. Although negative selection is usually associated with a BIS, there is actually some positive selection that goes on (from a certain perspective) with the T-cells. Since T-cells do not bind directly to antigens, but rather to MHCs (major histocompatibility complex) on the outside of cells that have attached themselves to antigens, T-cells must recognize the self MHCs. Therefore, T-cells go through positive selection. If T-cells do not match self MHCs, they are not allowed out of the thymus because they cannot perform their duty if they do not recognize self MHCs.

A.1.7 Negative Selection. Negative selection is the process by which autoimmune lymphocytes are purged from the repertoire. Negative selection happens with T-Cells and B-cells, the two types of lymphocytes. For T-Cells, negative selection happens in the thymus and on its periphery. The thymus contains many MHC-bearing antigen presenting cells (APCs). Negative selection takes place when immature T-Cells interact with these APCs. If a T-Cell recognizes a self-peptide MHC as presented by an APC, then the T-Cell is purged because it is potentially autoreactive and may attack self cells.

Negative selection for immature B-Cells takes place within the bone marrow. If an immature B-Cell recognizes a self cell in the bone marrow, then that B-Cell is removed from the inventory before the B-Cell enters the rest of the body and does harm by attacking self cells.

Appendix B. Random Number Generator

When executing any algorithm with a stochastic element, it is important to take into account the properties of the random number generator. A poor random number generator may induce unwanted (and unnoticed!) side effects. We desire a random number generator that is well tested founded in sound theory. We choose the Java random number generator that comes with the Sun SDK [86]. We used the `Random` class in the package `java.util`. The API specifications for this float random number generator says that the "polar method of G. E. P. Box, M. E. Muller, and G. Marsaglia, as described by Donald E. Knuth in The Art of Computer Programming, Volume 2: Seminumerical Algorithms, section 3.4.1, subsection C, algorithm P" is used for generating floats from a Gaussian distribution. A "linear congruential pseudorandom number generator, as defined by D. H. Lehmer and described by Donald E. Knuth in The Art of Computer Programming, Volume 2: Seminumerical Algorithms, section 3.2.1" is to generate random ints that are then used to generate random floats from a uniform distribution [85].

```
input : $e$: an ellipsoid to be inserted
input : $n$: the node that $e$ is to be inserted into
input : $p$: A set of random points that are inside $e$
output: The fraction of $n$ that $e$ covers that is not covered by a previously inserted ellipsoid
if There is already an ellipsoid that covers all of $n$ then
    return 0.00;

if $e$ covers all of $n$ because all corners of $n$ are in $e$ then
    add $e$ to a list of ellipsoids that cover all of $n$;
    return;

if $n$ is a leaf then
    if $p$ is not empty then
        return 1.00;
    else
        return 0.00;
    end

$sum = 0.00$;
forall children $c$ of $n$ do
    get a set $p_c \subseteq p$ such that all elements of $p_c$ are in $c$;
    $sum += insertEllipsoid(e, n, p_c)$;
end
return $\frac{sum}{|c_n|}$, where $c_n$ is the children of $n$;
```

Algorithm 1: *insertEllipsoid*

```
input  : $e$: an ellipse to be inserted
input  : $n$: the node that $e$ is to be inserted into
output: The fraction of $n$ that $e$ covers that is not covered by a previously inserted ellipse
$\mu_n \leftarrow$ The fraction of $n$ that is not covered by previously inserted ellipses.;
_$n \leftarrow$ The number of corners that $n$ has.;
$\ell_e \leftarrow$ The length of the longest semiaxis of $e$;
$\ell_n \leftarrow$ The length of the distance from the center of $n$ to a corner of $n$;
$childCount_n \leftarrow$ The number of children that $n$ has;
if $n$ is already totally covered by previously inserted ellipses. then
    return 0.00;

if $n$ is a leaf then
    if all corners of $n$ are inside of $e$ then
        $retVal = \mu_n$;
        $\mu_n = 0.00$;
        return $retVal$;

    else if $\alpha$ corners of $n$ are in $e$, where $\alpha < c_n$ then
        $retVal = \mu_n * \alpha / c_n$;
        $\mu_n = \mu_n - retVal$;
        return $retVal$;
    else if the center of $n$ is inside of $e$ then
        $retVal = 0.50 * \mu_n$;
        $\mu_n = \mu_n - retVal$;
        return $retVal$;
    else if the center of $e$ is inside of $n$ then
        $retVal = (1.00/c_n) * \mu_n$;
        $\mu_n = \mu_n - retVal$;
        return $retVal$;
    else
        return 0.00;

else
    if the euclidean distance between the centers of $n$ and $e$ is greater than $(\ell_e + \ell_n)$ then
        return 0.00

    else if all corners of $n$ are inside of $e$ then
        $retVal = \mu_n$;
        $\mu_n = 0.00$;
        return $retVal$;
    else
        $totalCoverage = 0.00$;
        forall child in n.children do
            $totalCoverage + = insertEllipsoid(e, child)$;
        end
        $retVal = totalCoverage/childCount_n$;
        $\mu_n = \mu_n - retVal$;
        return $retVal$;

end
```

Algorithm 2: *insertEllipsoid*

```
input  : $n$: number of ellipses in the population
input  : $genMax$: the maximum number of generations
input  : $dep$: depth of the $2^n$-way tree
input  : $mutStdDev$: The mutation standard deviation
input  : $childCount$: The number of children generated at each generation
input  : $nonMutGen$: Every $nonMutGen$ generations, members of the population are not mutated
input  : $mutChangeGens$: Every $mutCheckGens$, a check is made to decide if $mutStdDev$ should be
         decreased
output: $P$, a set of $n$ ellipses
2.1  for $curGen = 0; curGen < genMax; curGen++$ do
2.2      $children = generateEllipses(childCount)$;
2.3      $growEllipses(children)$;
2.4      $\beta = copy(P_{curGen})$ //deep copy of $P_{curGen}$;
2.5      $diff \leftarrow$ the index of the first difference between $P_{curGen}$ and $P_{curGen-1}$;
2.6      for $i \leftarrow diff; i < n; i++$ do
2.7          $P_{curGen}[i].objectiveVal = 0$;
2.8      end
2.9      if $curGen$ $mod$ $nonMutGen! = 0$ then
2.10         $mutate(P[i : diff - 1], mutStdDev)$;
2.11
2.12     $P_{curGen} = P_{curGen} \cup children$;
2.13     $sortByVolume(P_{curGen})$ //larger ellipses are first after sorting;
2.14     $clearSegmentTree()$;
2.15     for $i = 0; i < P_{curGen}.size(); i++$ do
2.16         $p = P_{curGen}[i]$;
2.17         $p.objectiveValue = insertEllipsoidIntoSegmentTree(p)$;
2.18         $p.penaltyValue =$
             $p.objectiveValue - (p.objectiveValue/((2.00 * numPointsCoveredInSegmentTree(p) + 1.00))$;
2.19     end
2.20     if $curGen$ $mod$ $nonMutGen! = 0$ then
2.21         for $i = 0; i < n; i++$ do
2.22             $\alpha =$ the index of the copy of $\beta[i]$ in $P_{curGen}$;
2.23             $p = P_{curGen}[\alpha]$;
2.24             if $(\beta[i].objectiveValue - \beta[i].penaltyValue) > (p.objectiveValue - p.penaltyValue)$ then
2.25                 $P_{curGen} = \beta[i]$;
2.26
2.27         end
2.28
2.29     $sortByObjectiveValueMinusPenaltyValue(P_{curGen})$;
2.30     $P_{curGen+1} = P_{curGen}[0 : n - 1]$;
2.31     if $curGen$ $mod$ $mutChangeGens! = 0$ then
2.32         $noChange = false$;
2.33         for $i = 0; i < n; i++$ do
2.34             $p = P_{curGen}[i]$;
2.35             $\alpha = P_{curGen-mutChangeGens}[i]$;
2.36             if $p.objectiveValue - p.penaltyValue > \alpha.objectiveValue - \alpha.penaltyValue$ then
2.37                 $noChange = true$;
2.38
2.39         end
2.40         if $noChange == false$ then
2.41             $mutStdDev* = 0.50$;
2.42
2.43
2.44 end
```

Algorithm 3: The complexity of $generateEllipses$ is $O(2^{d+d\alpha} * d^2 * genMax)$. The $O(2^{d+d\alpha} * d^2$ term is from the functions $insertEllipsoidIntoSegmentTree$ and $numPointsCoveredInSegmentTree$, while $genMax$ is the number of generations.

```
     input  : $S$: set of all self training points
     input  : $e$: An ellipse to be grown. This ellipse must be a sphere.
     output : $e$: radius of $e$ is maximized with the constraints that it covers no points in $s$ and its center is
              not changed.
3.1  forall $s \in S$ do
3.2      $smallestDist \leftarrow min(euclideantDistance(e.center, s), smallestDist)$;
3.3  end
```

Algorithm 4: The complexity of $growEllipse$ is $O(|S| * d)$, where d is the number of dimensions.

```
     input  : $e$: ellipse
     input  : $mutStdDev$: This affects how large the mutation can be.
     output : $e$: ellipse with mutated center.
4.1  foreach dimension $d$ do
4.2      $mutRange \leftarrow d.rangeSize/4.00$;
4.3      $mutVal \leftarrow randomGaussian() * mutRange * mutStdDev$;
4.4      $e.center[d] \leftarrow e.center[d] + mutVal$;
4.5  end
     /*Check that center is not outside of range. If it is, mutate it towards the range.   */
```

Algorithm 5: The complexity of $mutateCenter$ is $O(d)$.

```
     input  : $e$: ellipse
     input  : $mutStdDev$: This affects how large the mutation can be.
     output : $e$: ellipse with the length of one randomly chosen semiaxis mutated.
5.1  $d \leftarrow randomlyChooseDimension()$
     $changeVal \leftarrow randomGaussian() * e.semiaxes[d].length * mutStdDev$;
5.2  $e.semiaxes[d].length \leftarrow e.semiaxes[d].length + changeVal$;
     /*Check to see that the mutation did not make the length negative.   */
```

Algorithm 6: The complexity of $mutateLength$ is $O(1)$.

```
     input  : $e$: ellipse
     input  : $mutStdDev$: This affects how large the mutation can be.
     output: $e$: ellipse with the length of one randomly chosen semiaxis mutated.
     /*$s[0]$ and $s[1]$ must be different.                                          */
6.1  ;
6.2  $s[0] \leftarrow randomlyChooseSemiaxis(e)$;
6.3  $s[1] \leftarrow randomlyChooseSemiaxis(e)$;
6.4  $rAngle = randomAngle()4$;
     /*The following two lines are vector operations, requiring $d$ operations      */
6.5  $s[0] \leftarrow (cosine(rAngle) * s[0]) + (sine(rAngle) * s[1])$;
6.6  $s[1] \leftarrow (-1.00 * sine(rAngle) * s[1]) + (cosine(rAngle) * s[0])$;
     /*check to see that $s[0]$ and $s[1]$ are still within the dimension ranges    */
6.7  foreach chosen semixis $s$ do
6.8      foreach Dimension $d$ do
             /*Make sure that the $s$ is still within the boundaries for $d$.         */
6.9      end
6.10 end
```

Algorithm 7: The complexity of *mutateOrientation* is $O(d)$, where d is the number of dimensions.

```
input  : $e$: ellipse
input  : $stn$: $2^n$-way tree node
output: $fracCovered$: a value in [0.00, 1.00] that is the fraction of $s$ that $e$ gets credit for covering.
$d \leftarrow$ number of dimensions;
if $stn$ is a leaf then
    if $allCornersInside(e, stn)$ then
        $fracCovered \leftarrow stn.fractionUncovered$;
        $stn.fractionUncovered \leftarrow 0$;
        return $fracCovered$;

    $insideCount = numCornersInsideEllipse(e, stn)$;
    if $insideCount! = 0$ then
        $cornerCount \leftarrow 2^d$;
        $fracCovered \leftarrow stn.fractionUncovered * (insideCount/cornerCount)$;
        $stn.fractionUncovered \leftarrow stn.fractionUncovered - fracCovered$;

    if $e.mahalanobisDistance(stn.center) < 1.00$ then
        $fractionUncovered \leftarrow 0.50 * stn.fractionUncovered$;
        $stn.fractionUncovered \leftarrow stn.fractionUncovered - fractionCovered$;
        return $fractionUncovered$;

    if $stn.containsPoint(e.center)$ then
        $cornerCount \leftarrow 2^d$;
        $fractionCovered \leftarrow (1.00/cornerCount) * stn.fractionUncovered$;
        $stn.fractionUncovered \leftarrow stn.fractionUncovered - fractionCovered$;
        return $fractionUncovered$;

    return 0.00;
else
    if $euclideanDistanceBetweenCenters(e, stn) >$
    $e.lengthOfLongestSemiaxis + stn.centerToCornerDistance$ then
        return 0.00;

    if $e.allCornersInside(stn)$ then
        $fractionCovered \leftarrow stn.fractionUncovered$;
        $stn.fractionUncovered 0.00$;
        return $fractionCovered$;

    $totalCoverage = 0.00$;
    forall $c \in stn.children$ do
        $totalCoverage+ = insertEllipsoidIntoSegmentTree(e, c)$;
    end
    $fractionCovered = totalCoverage/stn.numChildren$;
    $stn.fractionUncovered = stn.fractionUncovered - fractionCovered$;
    return $fractionCovered$;
end
```

Algorithm 8: The worst case for the complexity of $insertEllipsoidIntoSegmentTree$ is $O(2^d * d^2 * (2^d)^\alpha) = O(2^{d+d\alpha} * d^2$, where α is the depth of the segment tree. The term $(2^d)^\alpha$ is the bound on the number of nodes in the segment tree of depth α in d dimensions. This is worst case scenario because an ellipse usually only ends up being tested against a small fraction of the total nodes, however this fraction is not known. Further analysis may be able to put a bound on the fraction of nodes that an ellipse is tested against against, but that is outside of scope of this research. The 2^d term is the number of corners in a node and the d^2 term is the complexity of the Mahalanobis distance, used to test whether a corner is inside of an ellipse.

```
     input  : $e$: ellipse
     input  : $stn$: $2^n$-way tree node
     output: $allInside$: A boolean, $true$ if all corners of $stn$ are inside of $e$, $false$ otherwise.
8.1  $allInside = true$;
8.2  forall $c \in stn.corners$ do
8.3      if $mahalanobisDistance(e, stn) > 1.00$ then
8.4          $allInside = false$;
8.5
8.6  end
8.7  return $allInside$;
```

Algorithm 9: Complexity of *allCornersInside* is $O(d^2 * 2^d)$, where d is the number of dimensions. The d^2 term is from Mahalanobis distance and the 2^d term is the number of corners in the segment tree node.

```
      input  : $e$: ellipse
      input  : $stn$: $2^n$-way tree node
      output: $insideCount$: The number of the points in $stn$ that are inside of $e$.
9.1   $d \leftarrow$ number of dimensions;
9.2   if $stn.pointCount == 0$ then
9.3       return 0;
9.4
9.5   if $euclideanDistanceBetweenCenters(e, stn) >$
      $e.lengthOfLongestSemiaxis + stn.centerToCornerDistance$ then
9.6       return 0.00;
9.7
9.8   $cornerCount = 2^d$;
9.9   if $cornerCount < stn.pointCount$ then
9.10      if $allCornersInside(e, stn)$ then
9.11          return $stn.pointCount$;
9.12
9.13
9.14  if $stn$ is leaf then
9.15      $insideCount \leftarrow 0$;
9.16      foreach $c \in stn.children$ do
9.17          $insideCount \leftarrow insideCount + numPointsCoveredInSegmentTree(e, c)$;
9.18      end
9.19      return $insideCount$;
9.20  else
9.21      $insideCount \leftarrow 0$;
9.22      foreach $p \in stn.pointCount$ do
9.23          if $mahalanobisDistance(e, p) < 1.00$ then
9.24              $insideCount \leftarrow insideCount + 1$;
9.25
9.26      end
9.27      return $insideCount$;
9.28  end
```

Algorithm 10: The worst case for the complexity of *numPointsCoveredInSegmentTree* is $O(2^d * d^2 * (2^d)^\alpha) = O(2^{d+d\alpha} * d^2$, where α is the depth of the segment tree. The term $(2^d)^\alpha$ is the bound on the number of nodes in the segment tree of depth α in d dimensions. This is worst case scenario because an ellipse usually only ends up being tested against a small fraction of the total nodes, however this fraction is not known. Further analysis may be able to put a bound on the fraction of nodes that an ellipse is tested against against, but that is outside of scope of this research. The 2^d term is the number of corners in a node and the d^2 term is the complexity of the Mahalanobis distance, used to test whether a corner is inside of an ellipse.

Bibliography

1. "Java Lapack". internet. Java Linear Algebra Library, http://www.cs.utk.edu/f2j/download.html.

2. "JAva MAtrix Package". internet. Java Matrix Package, http://math.nist.gov/javanumerics/jama/.

3. "Java Numerics". internet. Java Linear Algebra Library, http://math.nist.gov/javanumerics.

4. "Mahalanobis Distance". internet. Tutorial, http://www.wu-wien.ac.at/usr/h99c/h9951826/distance.pdf.

5. "McAfee Home". internet. Http://www.mcafee.com/us.

6. "Mealy Machine". internet. Tutorial, http://en.wikipedia.org/wiki/Mealy_machine.

7. "Panda Software: Virus and Intrusion Prevention for You PC". internet. Http://www.pandasoftware.com.

8. "Symantec Home". internet. Http://www.symantec.com.

9. "tcpdump". Program for filtering network traffic data. http://www.tcpdump.org/.

10. Ada, Gordon L. and Sir Gustav Nossal. "The Clonal-Selection Theory". *Scientific American*, 257:62–69, August 1987.

11. Aickelin, U., S. Cayzer, J. Kim, and J. McLeod. "Danger Theory: The Link between AIS and IDS?" *Proceedings of the 2nd International Conference on ARtificial Immune Systems*, 147–155. Springer-Verlag, 2003.

12. Aickelin, Uwe and Steve Cayzer. "The Danger Theory and Its Application to Artificial Immune Systems". *Proceedings of the 1st International Conference on Artificial Immune Systems*, 141–148. Morgan Kaufmann Publishers, Canterbury, UK, 2002.

13. Aickelin, Uwe, Julie Greensmith, and Jamie Twycross. "Immune System Approaches to Intrusion Detection - A Review". *Proceedings of International Conference on Artificial Immune Systems*, 316–329. 2004.

14. Amoroso, Edward. *Intrusion Detection.* Intrusion.Net Books, Sparta, New Jersey, 1999.

15. Anchor, Kevin P., Jesse B. Zydallis, Gregg H. Gunsch, and Gary B. Lamont. "Different Multi-Objective Evolutionary Programming Approaches for Detecting Computer Network Attacks". *Proceedings of Proceedings of Evolutionary and Multi-Objective Optimization 2003*, 707–721.

16. Balthrop, Justin, Fernando Esponda, Stephanie Forrest, and Matthew Glickman. "Coverage and Generalization in an Artificial Immune System". *Proceedings of the Genetic and Evolutionary Computation Converence*, 3–10. Morgan Kaufmann, New York, July 2002.

17. Bäck, T., D. B. Fogel, and T. Michalewicz (editors). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

18. Blake, C.L. and C.J. Merz. "UCI Repository of machine learning databases", 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Http://www.ics.uci.edu/∼mlearn/MLRepository.html.

19. de Castro, L. N. and F. J. Von Zuben. "The Clonal Selection Algorithm with Engineering Applications". *Proceedings of Genetic and Evolutionary Computation Conference*, 36–37. 2000.

20. de Castro, Leandro Nunes and Jonathan Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, London, England, first edition, 2002.

21. de Castro, Leandro Nunes and Fernando J. Von Zuben. "An Evolutionary Immune Network for Data Clustering". *Proceedings of the IEEE Brazilian Symposium on Artificial Neural Networks*, 84–89. 2000.

22. Center, Information Assurance Technology Analysis. *Intrusion Detection: Information Assurance Tools Report*. Technical report, Information Assurance Technology Analysis Center, 2001.

23. Dasgupta, Dipankar (editor). *Artificial Immune Systems and Their Applications*. Springer-Verlag, Berlin, 1999.

24. Dasgupta, Dipankar and Fabio Gonzalez. "An Immunity-Based Technique to Characterize Intrusions in Computer Networks". *IEEE Transactions on Evolutionary Computation*, 6(3), June 2002.

25. Dasgupta, Dipankar, Senhua Yu, and Nivedita Sumi Majumdar. "MILA-Multilevel Immune Learning Algorithm". *Proceedings of the Genetic and Evolutionary Algorithm Conference*. Springer, July 2003.

26. Denning, D.E. "An Intrusion-Detection Model". *IEEE Transactions on Software Engineering*, SE-13:222–232, 1987.

27. D'haeseleer, Patrik, Stephanie Forrest, and Paul Helman. "An Immunological Approach to Change Detection: Algorithms, Analysis and Implications". *Proceedings of the IEEE Symposium on Computer Security and Privacy*, 110–119. 1996.

28. Dozier, Gerry, Douglas Brown, John Hurley, and Krystal Cain. "Vulnerability Analysis of Immunity-Based Intrusion Detection Systems Using Evolutionary Hackers". *Proceedings of Genetic and Evolutionary Computation Conference*, 263–274. Seattle, WA, June 2004.

29. Duda, Richard O., Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, second edition, 2001.

30. Escamilla, Terry. *Intrusion Detection*. John Wiley & Sons, 1998.

31. Eskin, Eleazar. "Anomaly Detection over Noisy Data using Learned Probability Distributions". *Proceedings of the Seventeenth International Conference on Machine Learning*. JUNE 2000.

32. Forrest, Stephanie, Steven A. Hofmeyr, and Anil Somayaji.

33. Forrest, Stephanie, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. "A Sense of Self for Unix Processes". *Proceedings of the IEEE Symposium on Computer Security and Privacy*, 120–128. 1996.

34. Forrest, Stephanie, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. "Self-nonself Discrimination in a Computer". *Proceedings of the IEEE Symposium on Research in Security and Privacy*. 1994.

35. Forrest, Stephanie, Robert E. Smith, Brenda Javornik, and Alan S. Perelson. "Using Genetic Algorithms to Explore Pattern Recognition in the Immune System". *Evoutionary Computation*, 1:191–211, 1993.

36. Garrett, Simon M. "Parameter-Free, Adaptive Clonal Selection". *Proceedings of Congress on Evolutionary Computation (CEC 2004)*. Portland, OR, USA, JUNE 2004.

37. Goldberg, David E., Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. *Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms*. Technical report, University of Illinois at Urbana-Champagne.

38. Gomez, Jonatan, Fabio A. Gonzalez, and Dipankar Dasgupta. "An Immuno-Fuzzy Approach to Anomaly Detection". *Proceedings of The IEEE International Conference on Fuzzy Systems*. May 2003.

39. Gonzalez, F., D. Dasgupta, and J. Gomez. "The Effect of Binary Matching Rules in Negative Selection". *Proceedings of the Genetic and Evolutionary Algorithm Conference*. Springer, July 2003.

40. Gonzalez, Fabio A. and Dipankar Dasgupta. "Anomaly Detection Using Real-Valued Negative Selection". *Proceedings of Genetic Programming and Evolvable Machines*, 383–403. Kluwer Academic Publisher.

41. Gonzalez, Fabio A. and Dipankar Dasgupta. "An Immunogenetic Technique to Detect Anomalies in Network Traffic". *Proceedings of Genetic and Evolutionary Computation Conference*, 1081–1088. Morgan Kaufmann Publishers, July 2002.

42. Hamaker, Janna Shaffer. "Non-Euclidean Distance Measures in AIRS, an Artificial Immune Classification System". *Proceedings of Congress on Evolutionary Computation*. Portland, OR, USA, JUNE 2004.

43. Hang, Xiaoshu and Honghua Dai. "Constructing Detectors in Schema Complementary Space for Anomaly Detection". *Genetic and Evolutionary Computation Conference*. Springer-Verlag, Berlin.

44. Harmer, Paul K., Paul D. Williams, Gregg H. Gunsch, and Gary B. Lamont. "An Artificial Immune System Architecture for Computer Security Applications". *IEEE Transactions on Evolutionary Computation*, 6(3):252–280, 2002.

45. Hart, Emma and Peter Ross. "Studies on the Implications of Shape-Space Models for Idiotypic Networks". *Proceedings of International Conference on Artificial Immune Systems*, 413–426. 2004.

46. Hightower, Ron R., Stephanie Forrest, and Alan S. Perelson. "The Evolution of Emergent Organization in Immune System Gene Libraries". *Proceedings of the Sixth International Conference on Genetic Algorithms*, 344–350. Morgan Kaufmann, 1995.

47. Hofmeyr, Steven A., Stehpanie Forrest, and Anil Somayaji. "Intrusion Detection using Sequences of System Calls". *Journal of Computer Security*, 6:151–180, 1998.

48. Hou, Haiyu, Jun Zhu, and Gerry Dozier. "Artificial Immunity Using Constraint-Based Detectors", 2002.

49. Ji, Zhou and Dipankar Dasgupta. "Real-Valued Negative Selection Algorithm with Variable-Sized Detectors". *Proceedings of Genetic and Evolutionary Computation Conference*. Springer-Verlag, Berlin.

50. J.P., Anderson. *Computer Security Threat Monitoring and Surveillance*. Technical report, James P. Anderson Company, Fort Washington, PA.

51. Kelly, Patrick M., Don R. Hush, and James M. White. "An Adaptive Algorithm for Modifying Hyperellipsoidal Decision Surfaces". *Journal of Artificial Neural Networks*, 1:49–480, 1994.

52. Kephart, Jeffrey O. "A Biologically Inspired Immune System for Computers". *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 130–139. 1994.

53. Khoshgoftaar, Taghi M. and Mohamed E. Abushadi. "Resource-Sensitive Intrusion Detection Models for Network Traffic". *Proceedings of the Eighth International Symposium on High Assurance Systems Engineering*. Tampa, FL, March 2004.

54. Kim, J. and P. Bentley. "An Artificial Immune Model for Network Intrusion Detection". *Proceedings of the Seventh European Congress on Intelligent Techniques and Soft Computing*. Aachen, Germany, September 1999.

55. Kim, J. and P. Bentley. "The Human Immune System for Network Intrusion Detection". *Proceedings of the Seventh European Congress on Intelligent Techniques and Soft Computing*. Aachen, Germany, September 1999.

56. Kim, Jungwon and Peter J. Bentley. "An Evaluation of Negative Selection in an Artificial Immune System for Network Intrusion Detection".

57. Kim, Jungwon and Peter J. Bentley. "Towards an Artificial Immune System for Network Intrusion Detection: An Investigation of Dynamic Clonal Selection". *Proceedings of Congress on Evolutionary Computation*. Honolulu,.

58. Lamont, Gary B., Robert E. Marmelstein, and David A. Van Veldhuizen. *A Distributed Architecture for a Self-Adaptive Computer Virus Immune System*, chapter Eleven, 167–183. McGraw-Hill, UK, 1999.

59. Lane, Terran and Carla E. Brodley. "Temporal Sequence Learning and Data Reduction for Anomaly Detection". *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, 150–158. 1998.

60. Leder, Philip. "The Genetics of Antibody Diversity". *Scientific American*, 246:103–115, May 1982.

61. Lincoln Laboratory at Massachusetts Institute of Technology, http://www.ll.mit.edu/IST/ideval/data/1999/training/week1/index.html. *Lincoln Laboratory: DARPA Intrusion Detection Evaluation*.

62. Mahfoud, Samir. *Evolutionary Computation 1*, chapter 26: Boltzmann Selection, 195–200. Institute of Physics Publishing, 2000.

63. Mahoney, Matthew V. and Philip K. Chan. "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection", SEPTEMBER 2003.

64. Marmelstein, Robert E. *Evolving Compact Decision Set Rules*. Ph.D. thesis, Air Force Institute of Technology, WPAFB, Dayton, OH, 1999.

65. Matzinger, Polly. "The Real Function of the Immune System OR Tolerance and the Four D's (Danger, Death, Destruction and Distress)". internet.

66. McHale, John. "Sana Security Information Defense is Based on Human Immune System". *Military and Aerospace Electronics*, September 2004.

67. Mills, Robert F., Gilbert L. Peterson, and Wesley Allred. "Geometric Clustering for Intrusion Detection". To be submitted to the 2005 Conference on Recent Advances in Intrusion Detection (RAID 2005).

68. M.Mahoney and P. K. Chan. "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks". *Proceedings of Eleventh ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, 376–385. JULY 2002.

69. Mukherjee, Biswanath, L. Todd Heberlein, and Karl N. Levitt. "Network Intrusion Detection". *IEEE Network*, 8(3):26–41, 1994.

70. Ning, P. and Y. Cui. *An Intrusion Alert Correlator Based on Prerequisites of Intrusions*. Technical Report TR-2002-01, University of Illinois at Urbana-Champagne, 2001.

71. Ning, P., D.S. Reeves, and Y. Cui. *Correlating Alerts Using Prerequisites of Intrusions*. Technical Report TR-2001-13, North Carolina State University, Department of Computer Science, 1999.

72. Ning, Peng, Yun Cui, and Douglas S. Reeves. "Analyzing intensive intrusion alerts via correlation". *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection*. Zurich, Switzerland, October 2002. URL `citeseer.ist.psu.edu/ning02analyzing.html`.

73. Paeth, Alan W. (editor). *Graphics Gems V*. Academic Press, 1995. Kenneth J. Hill is the author of the chapter.

74. Peikari, Cyrus and Anton Chuvakin. *Security Warrior*. O'Reilly, 1 edition, 2004.

75. Peterson, Michael R., Travis E. Doom, and Michael L. Raymer. "GA-Facilitated Knowledge Discovery and Pattern Recognition Optimization Applied to the Biochem-

istry of Protein Solvation". *Proceedings Genetic and Evolutionary Computation Conference*, 426–437. Springer Verlag, 2004.

76. Potter, Mitchell A. and Kenneth A. De Jong. "The Coevolution of Antibodies for Concept Learning". *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, 530–539. Springer-Verlag, 1998. ISBN 3-540-65078-4.

77. Preparata, Franco P. and Michael Ian Shamos. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer-Verlag, 1985.

78. Sana Security, http://www.sanasecurity.com/. *Sana Security*.

79. Sana Security, http://www.sanasecurity.com/products/technology/index.php. *Sana Security: Technology*.

80. Sekar, R., M. Bendre, D. Dhurjati, and P. Bollineni. "A Fast Automaton-based Method for Detecting Anomalous Program Behaviors". *Proceedings of the IEEE Symposium on Security and Privacy*. 2001.

81. Smith, David J. and Mavina K. Vamanamurthy. "How Small is a Unit Ball". *Mathematics Magazine*, 62(2):103–107, April 1989.

82. Somayaji, A., S. Hofmeyer, and S. Forrest. "Principles of a Computer Immune System". *Proceedings New Security Paradigms*, 75–82. SEPTEMBER 1997.

83. Stepney, Susan, Robert E. Smith, Jonathan Timmis, and Andy M. Tyrrell. "Towards a Conceptual Framework for Artificial Immune Systems". *Proceedings of International Conference on Artificial Immune Systems*, 53–64. 2004.

84. Stibor, Thomas, Kpatscha M. Bayarou, and Claudia Eckert. "An Investigation of R-Chunk Detector Generation on Higher Alphabets". *Proceedings of Genetic and Evolutionary Computation Conference*, 299–307. 2004.

85. Sun, http://java.sun.com/reference/api/index.html. *Sun Java API Specifications*, September 2004.

86. Sun, http://www.java.sun.com. *Sun Java Technology Home Page*, September 2004.

87. Team, PAL Core Development. "PAL Project: Gamma Function", October 2004.

88. Tee, Garry J. "Ellipsoid Volume". personal communication, 2004. Received via email 10/29/2004.

89. Tee, Garry J. "Intersection of Ellipsoids". personal communication, 2004. Received via email 11/4/2004.

90. Tee, Garry J. *Surface Area and Capacity of Ellipsoids in n Dimensions*. Technical report, Department of Mathematics, University of Auckland, Auckland, New Zealand, March 2004.

91. Timmis, Jon and Mark Neal. "Investigating the Evolution and Stability of a Resource Limited Artificial Immune System". *Proceedings of the Genetic and Evolutionary Computation Conference*, 40–41. 2000.

92. Tonegawa, Susumu. “The Molecules of the Immune System”. *Scientific American*, 253:122–131, October 1985.

93. Wang, Ke and Salvatore J. Stolfo. “Anomalous Payload-based Network Intrusion Detection”. *Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection*. SEPTEMBER 2004.

94. Warrender, Christina, Stephanie Forrest, and Barak Pearlmutter. “Detecting Intrusions Using System Calls: Alternative Data Models”. *Proceedings of the IEEE Symposium on Security and Privacy*, 133–145. 1999.

95. Whitley, D., K. Mathias, S. Rana, and J. Dzubera. “Evaluating Evolutionary Algorithms”. *Artificial Intelligence*, 85:245–2761, 1996.

96. Williams, Paul, Kevin Anchor, John Bebo, Gregg Gunsch, and Gary Lamont. “Warthog: Towards a Computer Immune System for Detecting “Low and Slow“ Information System Attacks”. *Proceedings of Recent Advances In Intrusion Detection*. 2002.

97. Williams, Paul D. *Warthog: Toward an Artificial Immune system for Detecting ”Low and Slow” Information System Attacks*. Master’s thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, MARCH 2001.

98. Wolpert, David H. “No Free Lunch Theorems for Optimization”. *IEEE Transactions on Evolutionary Computation*, 1(1), April 1997.

99. Zhong, Shi, Taghi Khoshgoftaar, and Naeem Seliya. “Evaluating Clustering Techniques for Network Intrusion Detection”. *Proceedings of the* 10^{th} *ISSAT International Converence on Reliability and Quality Design*. Las Vegas, NV, August 2004.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 21–03–2005	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2003 — Mar 2005

4. TITLE AND SUBTITLE An Evolutionary Algorithm to Generate Ellipsoid Detectors for Negative Selection	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Shapiro, Joseph M., GS-09, NSA	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/05-20
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT Negative selection is a process from the biological immune system that can be applied to two-class (self and nonself) classification problems. Negative selection uses only one class (self) for training, which results in detectors for the other class (nonself). This paradigm is especially useful for problems in which only one class is available for training, such as network intrusion detection. Previous work has investigated hyper-rectangles and hyper-spheres as geometric detectors. This work proposes ellipsoids as geometric detectors. First, we establish a mathematical model for ellipsoids. We develop an algorithm to generate ellipsoids by training on only one class of data. Ellipsoid mutation operators, an objective function, and a convergence technique are described for the evolutionary algorithm that generates ellipsoid detectors. Testing on several data sets validates this approach by showing that our algorithm generates good ellipsoid detectors. Against artificial data sets, the detectors generated by our algorithm match $> 90\%$ of nonself data with 0% false alarm. Against a subset of data from the 1999 DARPA MIT intrusion detection data, the ellipsoids generated by our algorithm detect $\sim 98\%$ of nonself (intrusions) with a $\sim 0\%$ false alarm rate.

15. SUBJECT TERMS

Algorithms, Ellipsoids, Intrusion Detection, Geometry

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Gary B. Lamont, PhD, AFIT/ENG
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			
U	U	U	UU	158	**19b. TELEPHONE NUMBER** *(include area code)* (937) 785-3636, ext 4718