An Evolutionary Algorithm to Generate Hyper-Ellipsoid Detectors for Negative Selection

Joseph M. Shapiro, Gary B. Lamont, Gilbert L. Peterson
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
WPAFB, Dayton, Ohio 45433
{joseph.shapiro,gary.lamont,gilbert.peterson}@afit.edu

ABSTRACT

This paper introduces hyper-ellipsoids as an improvement to hyper-spheres as intrusion detectors in a negative selection problem within an artificial immune system. Since hyper-spheres are a specialization of hyper-ellipsoids, hyper-ellipsoids retain the benefits of hyper-spheres. However, hyper-ellipsoids are much more flexible, mostly in that they can be stretched and reoriented. The viability of using hyper-ellipsoids is established using several pedagogical problems. We conjecture that fewer hyper-ellipsoids than hyper-spheres are needed to achieve similar coverage of nonself space in a negative selection problem. Experimentation validates this conjecture. In pedagogical benchmark problems, the number of hyper-ellipsoids to achieve good results is significantly (∽50%) smaller than the associated number of hyper-spheres.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; E.1 [**DataStructures**]: Trees

General Terms

Algorithms

Keywords

Evolutionary computation, artificial immune systems, computational geometry, negative selection

1. INTRODUCTION

The human biological immune system distinguishes between self and nonself structures with amazing accuracy. Negative selection is a biological process by which the immune system generates nonself detectors that do not detect self structures. The biological negative selection process can be mapped to the computational domain as a two class pattern classification problem in an artificial immune system (AIS). Several efforts [9] [4] [5] provide a foundation for negative selection in an AIS for intrusion detection. This paper introduces an advance in the negative selection algorithm. Section 2 defines the problem. Section 3 describes and analyzes the design of the hyper-ellipsoid algorithm. Experimental design is set forth in Section 4. In Section 5 we analyze experimental results. Section 6 summarizes the paper.

Copyright 2005 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by a contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

2. BACKGROUND

This section contains a short summary of Artificial Immune Systems (AIS), a symbolic problem description and a discussion of detector generation using negative selection.

2.1 Artificial Immune System

An (AIS) is an element of the set of bio-inspired algorithms, which include evolutionary algorithms, neural networks, swarms, ant colony optimization, and others [14]. An artificial immune system is inspired by the human biological immune system (BIS). The BIS boasts an amazing ability to distinguish self (normally occurring, not harmful to body) from nonself (abnormal, often harmful to body). This is seen in the way that the BIS allows beneficial molecules/bacteria/etc. to work undisturbed while attacking invaders such as viruses. Negative selection is an important process that facilitates discrimination between self and nonself.

BIS Negative selection [2] occurs in the thymus, where new T-cells are semi-randomly generated. During negative selection, the BIS destroys T-cells that have an affinity for self structures. The result of negative selection is T-cells that have an affinity only for nonself invaders. After negative selection, the T-cells are released from the thymus into the body, where they take part in the search for nonself. In the computational domain, negative selection is mapped to a process which develops detectors (T-cells equivalents) for a class of data (nonself) by training only on the complement (self) of that class. Thus, the AIS generates detectors and then checks each detector to see if it matches any self points. If a detector matches any self points it is discarded. To summarize, negative selection as mapped to the computational domain is a two-class classification problem in which only one class is used for training.

2.2 Symbolic Problem Definition

A symbolic problem definition is important because it provides a mathematical description of the problem that is not ambiguous. Symbolically, the problem is formulated as follows: Let $F = \{f_1, f_2, ..., f_n\}$ be the set of n features in some data sample. Let P be the set of all possible points in feature space. Let $N \subseteq P$ be the set of all nonself points. Next, define the set of all self points: $S = P - N$. Let A be a set of antibodies. An antibody matches a subset of P. There is an isomorphism between an antibody $a \in A$ and the set of points it matches. We let $\Phi(a) \subseteq P$, so that $\Phi(a)$ is all of the points in P that the antibody a matches. A higher $|A|$ can provide better coverage of nonself space but results in computational overhead when testing data samples for membership in N. The goal is to maximize nonself coverage and minimize self coverage of A. These optimization criteria are described as:

$$\text{maximize } \left| \bigcup_{a \in A} \Phi(a) \right| \tag{1}$$

$$\text{minimize } \left| \left(\bigcup_{a \in A} \Phi(a) \right) \bigcap S \right| \tag{2}$$

2.3 Detector Generation

In the negative selection algorithm, 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, of course, not computationally efficient. If there are $|S|$ self points and $|N|$ nonself points and it is desired to create d good (does not match any self points) detectors then the time complexity increases exponentially in $|S|$ and linearly in d [9].

An implementation of negative selection requires a matching function for detectors. Let M be the matching function, so that $M : P \times A \to \{self, nonself\}$. Williams uses hyper-rectangles as a matching function in an AIS [17]. In the hyper-rectangle matching function, a detector is a hyper-rectangle. $M(p \in P, a)$ returns $nonself$ if p is inside of the hyper-rectangle associated with antibody a.

Dasgupta et. al. [3] suggest using Euclidean distance for a hyper-sphere detector. $a \in A$ matches $p \in P$ if p is inside of the hyper-sphere associated with a. They introduce an algorithm that generates hyper-sphere detectors. Dasgupta et. al. improve upon the hypersphere representation by using V-detectors, or "variable detectors" [4]. This differs from their original hyper-sphere model because they allow the detectors to have a variable radius. The flexibility of the variable radius allows the model to cover more nonself space with fewer detectors at no cost in computational complexity bounds.

McBride [6] uses an ellipsoid model in a classifier system. He clusters training points in the same class using a k-means algorithm. The covariance matrix for each cluster is computed and used to construct an ellipsoid around each cluster. McBride classifies test data by checking for membership in the ellipsoids. If a test point is not inside of an ellipsoid, it is classified according to the nearest ellipsoid. This approach may be termed a "positive selection" technique.

Our work builds in part upon the ellipsoid model of McBride and Tee and the hyper-sphere detectors of Dasgupta. Although McBride uses ellipsoids for classification, his ellipsoid foundation is not sufficient for this task, which is to generate a set of ellipsoids in nonself space. A unique contribution of this work is the development of a model for mutation of ellipsoids. Since hyper-spheres are a specialization of ellipsoids, it is hypothesized that ellipsoids perform better than hyper-spheres, since the flexibility of an ellipsoid allows it to fit more spaces than a hyper-sphere. "Better" means that ellipsoids can achieve the same nonself accuracy with fewer detectors. Ellipsoids are more complex than spheres, so the transition from spheres to ellipsoids is not trivial. The next section describes a model for ellipsoids and an evolutionary algorithm that generates ellipsoids to cover nonself space.

3. DESIGN

In this section we describe a design for an ellipsoid model and for an evolutionary algorithm that evolves a set of ellipsoids to cover nonself space.

3.1 Ellipsoid Model

An n-d ellipsoid is defined as follows:

$$(\mathbf{x} - \omega)^T \mathbf{A} (\mathbf{x} - \omega) = 1 \tag{3}$$

where $\mathbf{A}$ is a real symmetric positive-definite $n \times n$ matrix and ω, an $n \times 1$ matrix, is the center of the ellipsoid. Any vector $\mathbf{x}$ that satisfies Equation 3 is on the surface of the ellipse. In order to be viable for the current problem, the ellipsoid model must be able to provide either exact values or approximations for the volume of an ellipsoid and for whether or not $p \in P$ is inside of an ellipsoid. The following discussions answer these questions.

Volume of Ellipsoid

Let e be an n-d ellipsoid. Intuitively, an n-d ellipsoid is simply a sphere that has been stretched along the n orthogonal semiaxes of the ellipsoid. This indicates that the volume of an ellipsoid is simply the volume of a unit hyper-sphere multiplied by the length of each semiaxis. Tee (see Acknowledgements) proves that the volume of an n-dimensional ellipsoid is indeed

$$V = \Omega_n \ell_1 \ell_2 \cdots \ell_n \tag{4}$$

where Ω_n is the volume of an n-d hyper-sphere and $\ell_1, \ell_2, \ldots, \ell_n$ are the lengths of the n semiaxes of the ellipsoid. Smith and Vamanamurthy [12] show that the volume Ω_n of an n-dimensional unit hypersphere is calculable as

$$\Omega_n = \frac{\pi^{n/2}}{\Gamma(1 + \frac{1}{2}n)} \tag{5}$$

The Γ function is a mathematical extension of the factorial function from positive integers to real numbers. Weisstein [16] provides a definition and discussion of the Γ function. For implementation, we use Java source code for computing the Γ function from the PAL Project [15]. Evaluation of Equation 4 also requires $\ell_1, \ell_2, ...\ell_n$, the lengths of the semiaxes. The lengths of the semiaxes are easily derived from Equation 3. Since $\mathbf{A}$ is positive definite, it can be decomposed into the form $\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$, where $\mathbf{V}$ is a $n \times n$ matrix whose columns are orthonormal eigenvectors of $\mathbf{A}$ and $\mathbf{\Lambda}$ is a diagonal matrix whose diagonal entries are the eigenvalues associated with the eigenvectors in $\mathbf{V}$ [13]. Substituting for $\mathbf{A}$ in Equation 3 results in

$$(\mathbf{x}-\omega)^T \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T(\mathbf{x}-\omega)=1 \tag{6}$$

Further algebraic analysis shows that the diagonal entries in $\mathbf{\Lambda}$ are the inverses of the squares of the lengths of the semiaxes of the ellipse defined by Equation 3. That is, letting $\Lambda_{i,i}, 1 \le i \le n$ be the diagonals of Λ, then the length of the i^{th} semiaxis is defined by

$$\ell_i = 1/\sqrt{\Lambda_{i,i}} \tag{7}$$

Membership of a Point in an Ellipsoid

Let $\mathbf{p}$ be an n-d point and let e be an n-d ellipsoid as defined in Equation 3. Kelly et. al. [8] report that the Mahalanobis distance (left side of the inequality in Equation 8) can be used to determine whether or not $\mathbf{p}$ lies inside of e. $\mathbf{p}$ is inside of e if and only if the inequality in Equation 8 holds.

$$(\mathbf{p}-\omega)^T \mathbf{A}(\mathbf{p}-\omega) < 1 \tag{8}$$

3.2 Evolving a Set of Ellipsoids

With a good ellipsoid model, the next step is to produce a set of ellipsoids optimized according to Equations 1 and 2. An algorithm to solve this problem is unknown. Also, an exhaustive search for this problem has time complexity $O(\alpha^{2n+1}n^2)$ [11], where α is the number of discretized sections on each axis and n is the number of dimensions. For these reasons, we use an evolutionary algorithm (EA) to "evolve" good sets of ellipsoids. This section addresses the mapping of the ellipsoid model to representation, crossover, mutation, and objective function in the evolutionary algorithm domain.

Individual Representation

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, the cost of ellipsoid computations means that maintaining an entire population of sets of ellipsoids is a computational bottleneck. For this reason, we let an individual be one ellipsoid. The solution is all (or a subset) of the individuals in the population. The EA described in the following sections reflects this design decision.

Crossover With Ellipsoids

The chosen ellipsoid representation, however, does not lend itself to crossover. Crossover works well when an EA evolves a population of individuals, each representing a complete solution, and chooses the best individual when the algorithm finishes. In the ellipsoid detector generation problem, an individual could be a set of ellipsoids. Crossover is be performed by "trading" ellipsoids between different individuals. We do not use this pure EA method for the reasons cited in the previous paragraph.

Instead, we maintain only one set of ellipsoids. The final solution is a subset of the population 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 1 and 2 illustrate the problems with crossover when an individual does not represent a complete solution. As a result, crossover is not used in this research.

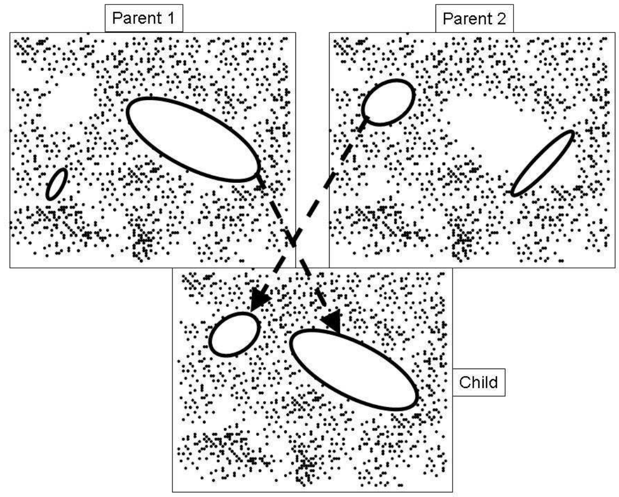


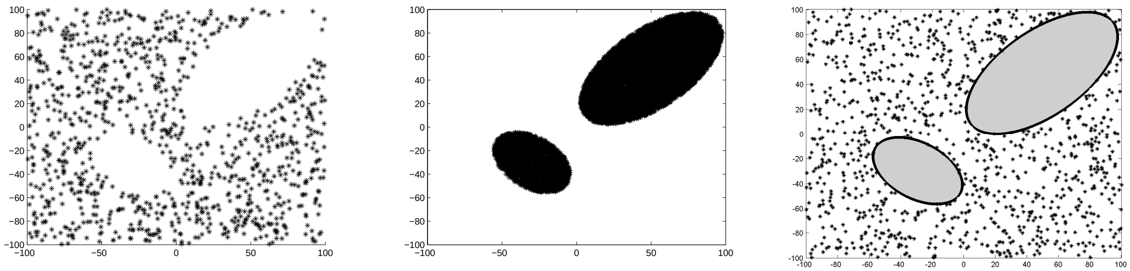
Figure 1: The two individuals on top are the parents. Each parent is a set of two ellipsoids. The obvious optimal solution to this problem is two ellipsoids. The arrows show how two "building blocks" (a "building block" is an ellipsoid) can be combined through crossover to produce a good child, which is the optimal solution.

Mutating an Ellipsoid

Evolutionary algorithms employ mutation as a variation operator that searches the solution space. There are two design goals for mutation. First, any valid ellipse can be mutated into any other valid ellipse through a finite series of mutations. 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.

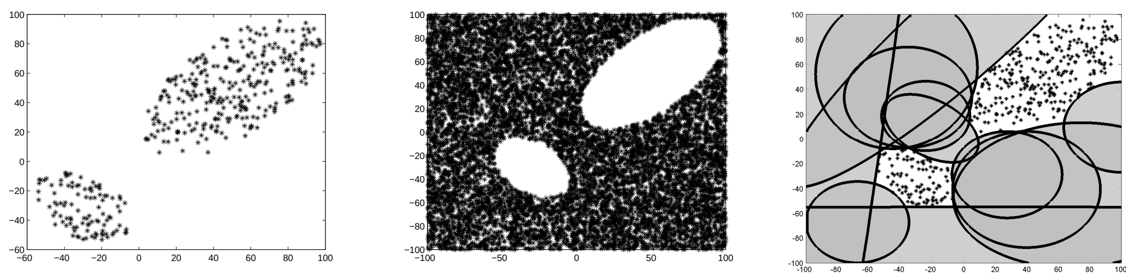
One method for mutating a ellipsoid might be to simply randomly change the covariance matrix. However, the covariance matrix is subject to the constraint that it is symmetric positive-definite and must remain so after mutation. The best way to guarantee that the covariance matrix remains symmetric positive-definite is to use the decomposition in Equation 6. This decomposition provides insight into three independent ways of mutating an ellipsoid: orientation by manipulating $\mathbf{V}$, center by manipulating ω, and semiaxis lengths by manipulating $\mathbf{\Lambda}$. In the following, let e be an ellipsoid defined by Equation 3.

Orientation Mutation: The orientation of an ellipsoid is the directions of the semiaxes. Shapiro [11] shows that the columns of $\mathbf{V}$ are the orthonormal semiaxes of e. Hence, the EA can mutate e by manipulating $\mathbf{V}$. The only constraint is that the EA mutation operator must return an $n \times n$ matrix with orthonormal columns.



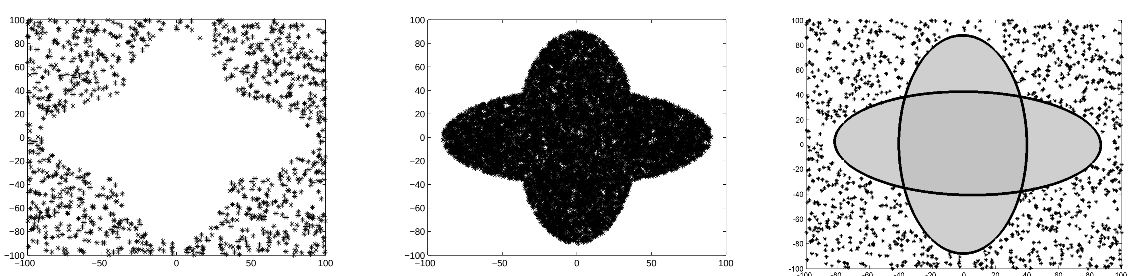
(a) self (b) nonself (c) results

Figure 3: Data set Val1. (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. (c) is the ellipsoids found by the ellipsoid algorithm.



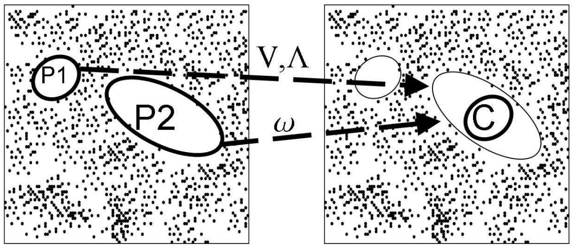
(a) self (b) nonself (c) results

Figure 4: Data set Val2. (a) has points inside of two ellipsoids. (b) is its associated test data set. (c) is the ellipsoids found by the ellipsoid algorithm.

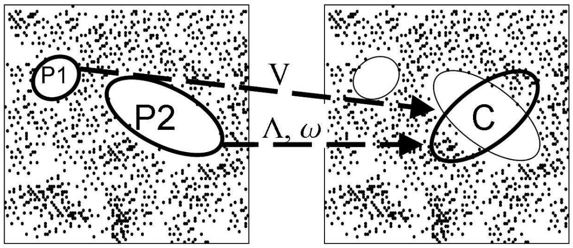


(a) self (b) nonself (c) results

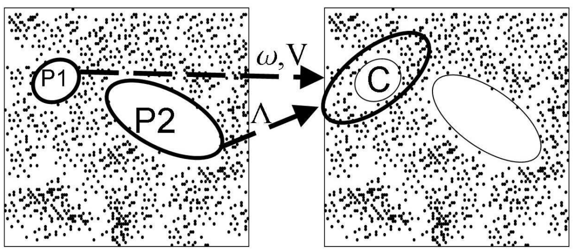
Figure 5: Data set Val3. In (a), the optimal solution is obviously two ellipsoids in a cross formation. (b) is its associated test data set. (c) is the ellipsoids found by the ellipsoid algorithm.



(a) Center



(b) Orientation



(c) Semiaxis Lengths

Figure 2: 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 6. $P1$ 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.

The EA accomplishes orientation mutation by rotating the ellipsoid in a 2d plane. Let $\mathbf{s}_1$ and $\mathbf{s}_2$ be the vectors of two semiaxes that are chosen at random from the n semiaxes of the ellipsoid. That means that $\mathbf{s}_1$ and $\mathbf{s}_2$ are two of the columns from the matrix $\mathbf{V}$ (see Equation 6). In the following discussion, it is important to note that $\mathbf{V}$ is orthonormal so $||\mathbf{s}_1|| = ||\mathbf{s}_2|| = 1$. A plane η is defined by all points that are a linear combination of $\mathbf{s}_1$ and $\mathbf{s}_2$. $\mathbf{s}_1$ and $\mathbf{s}_2$ are orthogonal to each other because they are semiaxes of an ellipse. $\mathbf{s}_1$ and $\mathbf{s}_2$ are mutated so that they stay in plane η and are still orthogonal to each other after the mutation. Let $\mathbf{s}_1^m$ and $\mathbf{s}_2^m$ be $\mathbf{s}_1$ and $\mathbf{s}_2$, respectively, after having undergone mutation. $\mathbf{s}_1^m$ and $\mathbf{s}_2^m$ are orthogonal to the $n-2$ unchosen semiaxes because the $n-2$ unchosen semiaxes are orthogonal to η and $\mathbf{s}_1^m$ and $\mathbf{s}_2^m$ are in η after the mutation.

This can also be shown mathematically. Let $\mathbf{s}_1^m = \alpha_1\mathbf{s}_1 + \beta_1\mathbf{s}_2$ and let $\mathbf{s}_2^m = \alpha_2\mathbf{s}_1 + \beta_2\mathbf{s}_2$ where α_i and β_i are coefficients in a linear combination. Let $\mathbf{u}$ be any one of the $n-2$ unchosen semiaxes. Two vectors are orthogonal if and only if their product is 0. $\mathbf{u}^T\mathbf{s}_1 = 0$ and $\mathbf{u}^T\mathbf{s}_2 = 0$ because $\mathbf{u}$ is orthogonal to s_1 and s_2. $\mathbf{u}$ is orthogonal to $\mathbf{s}_1^m$ and $\mathbf{s}_2^m$ because the dot products $\mathbf{u}^T\mathbf{s}_1^m$ and $\mathbf{u}^T\mathbf{s}_2^m$ are equal to 0.

To accomplish the rotation, a small angle θ is chosen from a Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = \frac{\pi}{2}$ radians. 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. As implemented, the orientation mutation has a parameter that increases or decreases the standard deviation. The EA computes s_1^m and s_2^m:

$$s_1^m = \cos(\theta)s_1 + \sin(\theta)s_2 \tag{9}$$

$$s_2^m = -\sin(\theta)s_1 + \cos(\theta)s_2 \tag{10}$$

The dot product of s_1^m and s_2^m is 0, (i.e.$((s_1^m)^T s_2^m) = 0$, so s_1^m and s_2^m are orthogonal to each other.

The described orientation mutation fulfills the two design goals. It is possible to achieve any orientation through a series of 2-d rotations and the mutation is small by virtue of the Gaussian distribution.

Center Mutation: The center point of e is ω. Hence, the EA center mutation operator must simply return any n-d vector. However, accomplishing the second design goal requires that the mutated center remain near the unmutated center.

The EA center mutation operator mutates each of the n components of ω individually. Let ω^m be ω after mutation. The center mutation operator chooses ω_i^m from a Gaussian distribution with mean $\mu = \omega_i$. The standard deviation for the Gaussian distribution is a parameter that can be changed.

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 the mutated center usually keeps the center point near its original location.

Semiaxis Length Mutation: The third type of mutation, semiaxis length, results when $\mathbf{\Lambda}$ is manipulated. The only constraint on the output of the semiaxis length mutation operator is that $\mathbf{\Lambda}^m$, the mutated $\mathbf{\Lambda}$, must be diagonal with positive entries. However, fulfillment of the second mutation design goal requires that the change in the lengths of the semiaxes should be relatively small.

The EA semiaxis length mutation operator mutates each of the n semiaxis lengths individually. Let ℓ_i^m be ℓ_i after mutation. The semiaxis length mutation operator chooses ℓ_i^m 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 $\ell_i^m > 0$.

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.

3.3 Objective Function

In an EA, the objective function evaluates the quality of the 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, which 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 a reward function and a 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. Let E be a population of ellipsoids. Let $REWARD : \{\pi | \pi \text{ is a set of ellipsoids } \} \times E \rightarrow [0.00, 1.00]$ be the reward function. $\sum_{e \in E} REWARD(E, e)$ 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.

We accomplish this in the following manner: First, E is sorted by volume using Equation 4. The algorithm then iterates through all of the ellipsoids to assign rewards, beginning with the largest volume. Each ellipsoid receives a reward for area that it covers and is not covered by any larger ellipsoid. The only remaining issue is how to compute how much area an ellipsoid covers that is not covered by a larger ellipsoid. To achieve this, we employ a 2^n-way tree data structure.

A 2^n-way tree [10, p.336-7] is a data structure that has its roots in computational geometry. A 2^n-way tree partitions a n-d space into 2^{nk} hyper-rectangles, where k is the number of levels of the tree. In this application, the algorithm rewards the ellipsoids by inserting them into a 2^n-way tree, largest ellipsoid first. Each node in the 2^n-way tree maintains a value $c \in [0, 1.00]$ 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. 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 [11].

Penalty Function

The penalty function discourages ellipsoids from covering self points. If an ellipsoid e covers β self points, its penalty function is

$$PENALTY(e) = 1.00 - (REWARD(E, e)/(2^{\beta} + 1)) \quad (11)$$

Evaluation of Equation 11 requires β, the number of self points that e covers. A naive approach for obtaining β is to check the ellipsoid against every point using the Mahalanobis distance, as described in Section 3.1. However, this quickly becomes a computational bottleneck as the number of points is increased. We employ a 2^n-way tree again. The algorithm inserts all of the self points into the 2^n-way tree so that each self point resides in a leaf node. Each node maintains z, the number of self points in its descendents. The algorithm obtains β by inserting e into the 2^n-way tree. Traversal is similar to the traversal described for the reward function. Heuristic tests are used so that e does not need to traverse every node in the 2^n-way tree. In this manner, the Mahalanobis distance is computed only for a subset of self points. Reference [11] provides full details of this algorithm.

We thus define the objective function,

$$OBJECTIVE(E, e) = REWARD(E, e) - PENALTY(e). \quad (12)$$

Note on Ellipsoid Comparison

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. We apply a statistical 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.

3.4 Algorithm Summary

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.

A detection generation algorithm must also determine the number of detectors to be used. This issue is left for future research. For the research described in this paper, the number of detectors is chosen a priori, based on problem domain knowledge and experience. This issue should be the first addressed in future research.

4. EXPERIMENTAL DESIGN

Experiments should test the ellipsoid model by answering these questions: Can an evolutionary algorithm evolve a set of ellipsoids to fill nonself space? How does the ellipsoid method compare with other geometric shapes, such as spheres? The above questions are first answered on pedagogical benchmark problems to validate the model. Then, a real world data set is used.

4.1 Pedagogical Data Sets

Pedagogical problems provide a proof of concept by validating that an algorithm produces expected results on problems with known characteristics. Such pedagogical problems also afford an opportunity for visualization techniques because they can be smaller and lower dimension. The intent is to generate a set of pedagogical self/nonself data sets that form various benchmark types based upon structural parameters. Such parameters include space dimensionality, space

Training Data	Algorithm	Detection Rate			False Alarm Rate			Detectors
		Mean	SD	P-Val	Mean	SD	P-Val	
Pedagogical 1	**Sphere**	93.40	2.77	0.00	0	0	1.00	16
	Ellipse	99.82	0.32		0	0		**2**
Pedagogical 2	**Sphere**	95.57	0.53	0.00	0	0	1.00	20
	Ellipse	97.53	0.28		0	0		**12**
Pedagogical 3	**Sphere**	94.39	1.57	0.01	0	0	1.00	20
	Ellipse	99.48	0.14		0	0		**2**
Setosa	**Sphere**	99.80	0.14	0.05	0.20	0.10	0.38	1
	Ellipse	99.20	0.34		1.20	0.24		**1**
Versicolor	**Sphere**	90.90	0.95	0.83	33.60	1.08	0.00	34
	Ellipse	90.80	1.00		17.00	0.83		**10**
Virginica	**Sphere**	98.00	0.40	0.87	33.80	1.06	0.00	36
	Ellipse	98.00	0.43		30.00	0.90		**20**

Table 1: Comparison of spherical and elliptical detectors. P-Val is the p-value, computed using a two-sided t-test to determine whether the ellipsoid and sphere samples could have the same mean.

geometry (ellipsoid, sphere, rectangle, etc.) and test point density.

The three artificial data sets selected are referred to as Val1, Val2 and Val3. Figures 3 and 5 present Val1 and Val3, two self data sets for which the optimal solution is two ellipses. Val3 tests whether the algorithm can find an optimal solution when overlapping ellipsoids are required. Val2, presented in Figure 4, is an inverse problem. It tests how well the algorithm can find a set of ellipsoids to fill in a space that is not elliptically shaped. Although the optimal solution is not known for Val2, a visual inspection of the results and analysis of test data provide a good approximation as to how well the algorithm performs.

After the algorithm has generated ellipsoids for Val1-Val3, a set of nonself points is needed for testing. We produce test data by generating random points in the inverse of the self area in each data set. We generate a large number (10,000) of test points for each data set so that test results accurately reflect the performance of the generated ellipsoids. Part (b) of Figures 3 - 5 shows that 10,000 points provides good coverage of the nonself region. The algorithm is run ten times on each dataset for statistical analysis.

4.2 Real-World Test Data Sets

We select a known real-world data set to continue algorithm validation because it is used by other researchers. Note that we are not attempting to determine the best features for classification, but only to reduce the error of classification (detection rate, false alarm rate), given the best features. The iris data set is obtained from the University of California at Irvine Machine Learning Repository [1]. The iris database originates from a classic taxonomy paper describing three different types of iris plants: Iris-Setosa, Iris-Versicolour, 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. Each instance has four real-valued attributes: sepal length, sepal width, petal length, and petal width. Of the three classes, one class is linearly separable from the other two, but the other two are not linearly separable from each other. The iris data set is well-suited for validation because it is moderately sized in dimensionality (four features) and data cardinality (50 points in each class). We use 90/10 cross-fold validation [7] for testing. In 90/10 cross fold validation, the data from the train class is randomly ordered and then divided into 10 equally sized sets. The two test classes are also randomly reordered and divided into 10 equally sized sets. 10 tests are performed. For the i^{th} test the algorithm trains on the complement of the i^{th} subset of the train class while the i^{th} subsets of R, J, and K are used for testing.

5. RESULTS AND ANALYSIS

Subfigure (c) in Figures 3-5 shows the results of running the ellipsoid algorithm against the corresponding self data sets. From a visual perspective, the algorithm is very successful. It finds the known solutions, covers non-elliptically shaped nonself space well, and even finds the optimal solution when it requires overlapping. When tested against the nonself test data shown in subfigure (b) of Figures 3-5, the algorithm also performs successfully. The ellipsoid algorithm covers all of the nonself test points in Figure 3 and about 95% of the nonself test points in Figures 4 and 5. These results are impressive, especially in Figure 4, since the ellipsoids must cover and area in the shape of inverted ellipsoids. The detection rate P Values in Table 1 show that the ellipsoids attain a better detection rate with fewer ellipsoids for the examples.

Table 1 shows the results for the Iris experiments. When training on class Setosa, the ellipsoids and spheres perform similarly. This result is expected because Setosa is linearly separable from the other two classes. The results are more interesting when training on Versicolor and Virginica. We expect the ellipsoids to achieve comparable performance with the spheres, but with a smaller number of detectors. This is because Versicolor and Virginica are not linearly separable. Hence, filling nonself space requires more flexibility, which is a strength of ellipsoids when compared to spheres. Table 1 shows the strength of ellipsoid model, which uses 70% and 44% less detectors than the sphere model for Versicolor and Virginica, respectively.

6. CONCLUSION

If AIS is to be applied to a detection problem, it is important to generate accurate and efficient detectors. Accurate means that the detectors correctly differentiate self from nonself. Efficient means that there are few detectors, since fewer detectors means less computation to decide whether test points are self or nonself (a test point must be compared against all detectors until it matches one). Ellipsoids generated by an evolutionary algorithm are a considerable improvement over other geometric shapes that have been previously investigated. They are more efficient than spheres and they achieve similar accuracy. Note that the ellipsoid algorithm has also achieved good results (92% detection, 0% false alarm rate) using the MIT intrusion data [11]. Future work may investigate crossover as a variation operator in the EA, additional mutation operators/parameters, the problem of over-training, and ways to improve upon the computational complexity of the ellipsoid model. Also, testing on additional pedagogical data sets and real world data applications is important to further validate our approach.

7. ACKNOWLEDGMENTS

Dr. Garry J. Tee of the Department of Mathematics, University of Auckland in Auckland, New Zealand provided a good starting point for the mathematics behind ellipsoids (tee@math.auckland.ac.nz).

This work represents the views of the authors and does not represent the views of the U.S. government or U.S. Air Force.

8. REFERENCES

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

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

[3] Fabio A. Gonzalez and Dipankar Dasgupta. Anomaly detection using real-valued negative selection. In *Proceedings of Genetic Programming and Evolvable Machines 2003*, pages 383–403. Kluwer Academic Publisher.

[4] Zhou Ji and Dipankar Dasgupta. Real-valued negative selection algorithm with variable-sized detectors. In *Proceedings of GECCO 2004, LNCS 3102*, Berlin. Springer-Verlag.

[5] Jungwon Kim and Peter J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, September.

[6] Brent McBride. A hyper-geometric data classifier for blind detection of novel steganography. Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 2004.

[7] D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors. *Machine, Neural and Statistical Classification*. Ellis Horwood, 1994.

[8] Don R. Hush Patrick M. Kelly and James M. White. An adaptive algorithm for modifying hyperellipsoidal decision surfaces. *Journal of Artificial Neural Networks*, 1:49–480, 1994.

[9] Stephanie Forrest Patrik D'haeseleer and Paul Helman. An immunological approach to change detection: Algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, pages 110–119, 1996.

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

[11] Joseph M. Shapiro. An evolutionary algorithm to generate hyper-ellipsoid detectors for negative selection. Master's thesis, Air Force Institute of Technology, Wright Patterson Air Force Base, Ohio, 2005.

[12] David J. Smith and Mavina K. Vamanamurthy. How small is a unit ball. *Mathematics Magazine*, 62(2):103–107, April 1989.

[13] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 2 edition, 1998.

[14] Jonathan Timmis Susan Stepney, Robert E. Smith and Andy M. Tyrrell. Towards a conceptual framework for artificial immune systems. In *Proceedings of International Conference on Artificial Immune Systems (ICARIS 2004)*, pages 53–64, 2004.

[15] PAL Core Development Team. Pal project: Gamma function, October 2004.

[16] Eric W. Weisstein. Gamma function. From Mathworld–A Wolfram Web Resource, http://mathworld.wolfram.com/GammaFunction.html.

[17] Paul Williams, Kevin Anchor, John Bebo, Gregg Gunsch, and Gary Lamont. Cdis-towards a computer immune system for detecting network intrusions. In *Proceedings of Recent Advances In Intrusion Detection*, 2002.