Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm

Andrew Watkins* and Jon Timmis
(abw5,jt6@kent.ac.uk)
Computing Laboratory, University of Kent. CT2 7NF. UK.

Lois Boggess
(lboggess@cse.msstate.edu)
Department of Computer Science and Engineering, Mississippi State University, USA

Abstract. This paper presents the inception and subsequent revisions of an immune-inspired supervised learning algorithm, Artificial Immune Recognition System (AIRS). It presents the immunological components that inspired the algorithm and describes the initial algorithm in detail. The discussion then moves to revisions of the basic algorithm that remove certain unnecessary complications of the original version. Experimental results for both versions of the algorithm and are discussed and these results indicate that the revisions to the algorithm do not sacrifice accuracy while increasing the data reduction capabilities of AIRS.

Keywords: supervised learning, artificial immune systems, classification, neural networks

1. Introduction

In recent years there has been considerable interest in exploring and exploiting the potential of Artificial Immune Systems for applications in computer science and engineering. These systems are inspired by various aspects of the immune systems of mammals. Some of these aspects, such as the distinction between self and non-self and the concept of negative selection, have a natural and intuitive fit for applications involving computer security, network intrusion detection [16], [20], [23], change detection, and the like. Moreover, research into natural immune systems suggests the existence of learning properties which may be used to advantage in machine learning systems [2]. With the exception of [7], until very recently Artificial Immune System (AIS) research into machine learning has focused on the development of unsupervised learning and clustering [10], [29] rather than supervised learning and reinforcement learning[1]. In contrast, this paper presents an AIS classifier which

* Also, Department of Computer Science and Engineering, Mississippi State University, USA.

[1] While others have discussed the similarity of Artificial Immune Networks and other connectionist models such as Artificial Neural Netorks [15], there seems to have

initially was intended to show that a subset of metaphors from the AIS literature could produce an effective supervised learning system [31]. The classifier that resulted performs well on a variety of publicly available test problems used by the machine learning community [32] and has been the object of study both to further refine its algorithms and to better understand the source of its classification power. This paper describes the original algorithm developed in [31] and proposes a modification which enhances the efficiency of the resulting classifier while retaining classification accuracy. An empirical review of the effects of this modification is presented. This paper is an expanded version of [34].

2. Bio-Inspired Computing and Machine Learning

The means of constructing and developing machine learning systems have been extremely diverse. These have ranged from the manipulation of symbolic data in order to develop a concept learning system to the heavy use of sub-symbolic representations of data in the development of function approximation algorithms (e.g., back-propagation neural networks). There are several good general surveys of the field of machine learning and its techniques (foremost being [26]), and no attempt will be made to duplicate that effort here. However, instead, the focus will be on one particular emerging field of computing science that can, perhaps, lend insight into the development of machine learning systems.

Since there seems to be a desire to build computational systems that exhibit some (if not all) of human cognitive abilities, one of the first clear examples of this has been in the proliferation of the field of artificial neural networks (ANNs). This field, as can be surmised from its name, has looked to the workings of the human brain as inspiration for the development of computing systems. Since these workings are not completely understood and since the goals of building our intelligent systems are often problem-driven, ANNs have by no means been an attempt to directly model all of the processes occurring within human brains. Instead, however, certain observed phenomena have been simplified and encoded in order to replicate some of the mechanisms of our thought processes that might best be suited to manipulating data and learning general patterns or trends in this data. Of course, pattern recognition has not been the only application of ANN techniques. They have shown to be successful for a wide variety of learning and control problems [3], [19].

been no direct translation of these ideas to a well-tested immune-inspired *supervised* learning system.

A second, obvious example, of a computing paradigm inspired by observations of natural phenomena is that of Genetic Algorithms (GAs). Looking to neo-darwinian evolutionary theory with an emphasis on reproduction, mutation, and genetic crossover, GAs have been successful in solving certain difficult or computationally expensive optimization problems. Darwinian evolutionary theories have also inspired other successful evolutionary programming techniques [25]. While these two examples have been the most prolifically discussed computational ideas inspired by natural observations, they are not the only ones. In recent years, we have seen the exploration of other metaphors from nature as applied to computing problems. These include swarming insects [22], ant colonies [5], and mammalian immune systems [9]. It is this latest source of inspiration, mammalian immune systems, and the use of this natural system as a guide to developing machine learning systems that will be the particular focus of this article.

2.1. The Artificial Immune Recognition System

AIRS (Artificial Immune Recognition System) is a novel immune inspired supervised learning algorithm [31]. Motivation for this work came from the author's identification of the fact that there was a significant lack of research that explored the use of the immune system metaphor for supervised learning; indeed, the only work identified was that of [7]. However, it was noted that within the AIS community there had been a number of investigations on exploiting immune mechanisms for unsupervised learning (that is, where the class of data is unknown *apriori*) [30], [29] and [11]). Work in [10] examined the role of the clonal selection process within the immune system [6] and went on to develop an unsupervised learning known as CLONALG. This work was extended by employing the metaphor of the immune network theory [21] and then applied to data clustering. This led to the development of the aiNet algorithm [11]. Experimentation with the aiNet algorithm revealed that evolved artificial immune networks, when combined with traditional statistical analysis tools, were very effective at extracting interesting and useful clusters from data sets. aiNet was further extended to multimodal optimization tasks [8]. Other work in [30] also utilized the immune network theory metaphor for unsupervised learning, and then augmented the work with the development of a resource limited artificial immune network [29], which reported good benchmark results for cluster extraction and exploration with artificial immune networks. Indeed, this work has been further extended by [27] with the introduction of fuzzy logic and refinement of various calculations. The work in [29] was of particular relevance to [31] and the further work described in

this paper, which builds on this previous work, in particular the ideas of artificial recognition balls and resource limitation from [29] and long-lived memory cells from [11]. However, while these population control mechanisms and data representation concepts were borrowed from this work on immune networks, it should be stressed that AIRS is in no way an immune **network** model of compuation. The rest of this section and describes the immune metaphors that have been employed within AIRS.

2.2. IMMUNE PRINCIPLES EMPLOYED

A little time should be taken to draw attention to the most relevant aspects of immunology that have been utilized as inspiration for this work. A more detailed overview of the immune system and its relationship with computer science and engineering can be found in [9]. Throughout a person's lifetime, the body is exposed to a huge variety of pathogenic (potentially harmful) material. The immune system contains lymphocyte cells known as B- and T-cells, each of which has a unique type of molecular receptor (location in a shape space). Receptors in this shape space allow for the binding of the pathogenic material (antigens), with higher affinity (complementarity) between the receptor and antigen indicating a stronger bind. Work in [9] adopted the term shape-space to describe the shape of the data being used, and defined a number of affinity measures, such as Euclidean distance, which can be used to determine the interaction between elements in the AIS. Within AIRS (and most AIS techniques) the idea of antigen/antibody binding is employed and is known as antigenic presentation. When dealing with learning algorithms, this is used to implement the idea of matching between training data (antigens) and potential solutions (B-Cells). Work in [29] employed the idea of an artificial recognition ball (ARB), which was inspired by work in [14] describing antigenic interaction within an immune network. Simply put, an ARB can be thought to represent a number of identical B-Cells and is a mechanism employed to reduce duplication and dictate survival within the population. Once the affinity between a B-Cell and an antigen has been determined, the B-Cell involved transforms into a plasma cell and experiences clonal expansion. During the process of clonal expansion, the B-Cell undergoes rapid proliferation (cloning) in proportion to how well it matches the antigen. This response is antigen specific. These clones then go through affinity maturation, where some undertake somatic hypermutation (mutation here is inversely proportional to antigenic affinity) and eventually will go through a selection process through which a given cell may become a memory cell. These memory cells

Table I. Mapping between the Immune System and AIRS

Immune System	AIRS
Antibody	Feature Vector
Recognition Ball	Combination of feature vector and vector class
Shape-Space	Type and possible values of the data vector
Clonal Expansion	Reproduction of ARBs that are well matched with antigens
Antigens	Training data
Affinity Maturation	Random mutation of ARB and removal of the least stimulated ARBs
Immune Memory	Memory set of mutated ARBs
Metadynamics	Continual removal and creation of ARBs and memory cells

are retained to allow for a faster response to the same, or similar, antigen should the host become re-infected This faster response rate is known as the secondary immune response. Within AIRS, the idea of clonal expansion and affinity maturation are employed to encourage the generation of potential memory cells. These memory cells are later used for classification. Drawing on work from [29], AIRS utilized the idea of a stimulation level for an ARB, which, again, was derived from the equations for an immune network described in [14]. Although AIRS was inspired by this work on immune networks, the development of the classifier led to the abandoning of the network principles in favor of a simple population-based model. In AIRS, ARBs experience a form of clonal expansion after being presented with training data (analogous to antigens); details of this process are provided in section 4.3. However, AIRS did not take into account the principle of affinity proportional mutation. When new ARBs were created, they were subjected to a process of random mutation with a certain probability and were then incorporated into the memory set of cells should their affinity have met certain criteria. Within the AIRS system, ARBs competed for survival based on the idea of a resource limited system [29]. A predefined number of resources existed, for which ARBs competed based on their stimulation level: the higher the stimulation value of an ARB the more resources it could claim. ARBs that could not successfully compete for resources were removed from the system. The term metadynamics of the immune system refers to the constant changing of the B-Cell population through cell proliferation and death. This was present in AIRS with the continual production and removal of ARBs from the population. Table I summarizes the mapping between the immune system and AIRS.

3. Overview of the AIRS algorithm

This section presents an overview of the AIRS algorithm. 3.1 defines some of the key terms and concepts important to the understanding of the algorithm. 3.2 provides a somewhat formal tour of the training routine of the algorithm. 3.3 presents a more conceptual overview of the algorithm.

3.1. Definitions

This section presents definitions of the key terms and concepts used throughout the rest of this paper, particularly as they apply to the AIRS algorithm.

- *affinity*: a measure of "closeness" or similarity between two *antibodies* or *antigens*. In the current implementation, this value is guaranteed to be between 0 and 1 inclusively and is calculated simply as the Euclidean distance of the two objects' *feature vectors*. Thus, small affinity values indicate strong affinity.
- *affinity threshold (AT)*: the average *affinity* value among all of the *antigens* in the *training set* or among a selected subset of these training antigens.
- *affinity threshold scalar (ATS)*: a value between 0 and 1 that, when multiplied by the *affinity threshold*, provides a cut-off value for memory cell replacement in the *AIRS* training routine.
- *antibody*: a *feature vector* coupled with its associated *output* or *class*; the feature vector-output combination is referred to as an antibody when it is part of an *ARB* or *memory cell*.
- *antigen*: this is the same in representation as an *antibody*; however, the feature vector-class combination is referred to as an antigen when it is being presented to the *ARBS* for stimulation and/or response.
- *Artificial Immune Recognition System (AIRS)*: a classification algorithm inspired by natural immune systems.
- *Artificial Recognition Ball (ARB)*: also known as a *B-Cell*. It consists of an *antibody*, a count of the number of *resources* held by the cell, and the current *stimulation value* of the cell.
- *B Cell*: in this paper, more commonly referred to as an *Artificial Recognition Ball*.

- *candidate Memory Cell*: the *antibody* of an *ARB*, of the same *class* as the training *antigen*, which was the most stimulated after exposure to the given antigen.

- *class*: the category of a given *feature vector*. This is also referred to as the *output* of a cell.

- *clonal rate*: an integer value used to determine the number of mutated clones a given *ARB* is allowed to attempt to produce. In the current implementation, a selected ARB is allowed to produce up to (*clonal rate* * *stimulation value*) mutated clones after responding to a given *antigen*. This product is also used in assigning *resources* to an ARB. Therefore, the clonal rate serves a dual-role as resource allocation factor and clonal mutation factor for the cell population.

- *established Memory Cell*: the *antibody* of an *ARB* which has survived competition for resources and was the most stimulated to a given training *antigen* and has been added to the evolving set of *memory cells*.

- *feature vector*: one instance of data represented as a sequence of values. Each position in the sequence represents a different feature associated with the data, and each feature has its own range of legitimate values.

- *hyper-mutation rate*: an integer value used to determine the number of mutated clones a given *memory cell* is allowed to inject into the cell population. In the current implementation, the selected memory cell injects at least (*hyper-mutation rate* * *clonal rate* * *stimulation value*) mutated clones into the cell population at the time of *antigen* introduction.

- *k nearest neighbor (KNN)*: a classification scheme in which the response of the classifier to a previously unseen item is determined by a majority vote among the k closest data points. For the *AIRS* algorithm, the k closest data points are in actuality the k most stimulated *memory cells* to a given test *antigen*.

- *k value*: the parameter which indicates how many *memory cells* should be used to determine the classification of a given test item. (see *k nearest neighbor* for more details)

- *memory cell (mc)*: the *antibody* of an *ARB* which was the most stimulated by a given training *antigen* at the end of exposure to

that antigen. It is used for hyper-mutation in response to incoming training antigens (see *hyper-mutation rate*). An mc can be replaced, however. This occurs only when a *candidate mc* is more stimulated to a given training antigen than the most stimulated *established mc* and the affinity between the established mc and the candidate mc is less than the product of the *Affinity Threshold* and the *Affinity Threshold Scalar*.

- *mutation rate*: a parameter between 0 and 1 that indicates the probability that any given feature (or the output) of an *ARB* will be mutated.

- *output*: the classification category associated with a cell. Same as the *class* of the feature vector corresponding to the cell.

- *resources*: a parameter which limits the number of *ARB*s allowed in the system. Each ARB is allocated a number of resources based on its *stimulation value* and the *clonal rate*. The total number of system wide resources is set to a certain limit. If more resources are consumed than are allowed to exist in the system, then resources are removed from the least stimulated ARBs until the number of resources in the system returns to the number allowed. If all of a given ARB's resources are removed, then that ARB is removed from the cell population.

- *seed cell*: an *antibody*, drawn from the *training set*, used to initialize *Memory Cell* and *ARB* populations at the beginning of training.

- *stimulation function*: a function used to measure the response of an *ARB* to an *antigen* or to another ARB. In the current formulation of the *AIRS* classifier, this function should return a value between 0 and 1 inclusively. For the implementation of *AIRS* presented in this study, the stimulation function is inversely proportional to the Euclidean distance between the *feature vector*s of the *ARB* and the *antigen*.

- *stimulation value*: the value returned by the *stimulation function*.

- *stimulation threshold*: a parameter between 0 and 1 used as a stopping criterion for the training on a specific *antigen*. For the current implementation, only when the average *stimulation value* of the *ARB*s of each class is above the stimulation threshold does training in reaction to the particular antigen stop.

- *test set*: the collection of *antigens* used to evaluate the classification performance of the trained *AIRS* classifier.

– *training set*: the collection of *antigens* used to train the *AIRS* classifier.

3.2. Tour of the Algorithm

This subsection presents a tour of the AIRS algorithm. In particular, this section presents an overview of the primary routines, methods, and equations used in the training and building of an immune-system based classifier. This somewhat formal view of the algorithm is offered as a companion to the more conceptual discussion and explanation of these mechanisms presented in 3.3. There are four primary stages involved in the AIRS algorithm. The first stage is data normalization and initialization. The second stage is memory cell identification and ARB generation. The third stage is competition for resources in the development of a candidate memory cell. The final stage of the training algorithm is the potential introduction of the candidate memory cell into the set of established memory cells.

For this discussion, let us establish the following notational conventions:

– Let MC represent the set of memory cells and mc represent an individual member of this set.

– Let $ag.c$ represent the class of a given antigen, ag, where $ag.c \in C = \{1, 2, \ldots nc\}$ and nc is the number of classes in the data set.

– Let $mc.c$ represent the class of a given memory cell, mc, where $mc.c \in C = \{1, 2, \ldots nc\}$.

– Define $MC_c \subseteq MC = \{MC_1 \cup MC_2 \cup \ldots MC_{nc}\}$ and $mc \in MC_c$ **iff** $mc.c \equiv c$.

– Let $ag.f$ and $mc.f$ represent the feature vector of a given antigen and memory cell, ag and mc, respectively. Let $ag.f_i$ represent the value of the ith feature in $ag.f$ and $mc.f_i$ the value of the ith value of $mc.f$.

– Let AB represent the set of ARBs, or the population of existing cells, and MU represent a set of mutated clones of ARBs. Furthermore, let ab represent a single ARB where $ab \in AB$.

– Let $ab.c$ represent the class of a given ARB, ab, where $ab.c \in C = \{1, 2, \ldots nc\}$.

– Define $AB_c \subseteq AB = \{AB_1 \cup AB_2 \cup \ldots AB_{nc}\}$, and $ab \in AB_c$ iff $ab.c \equiv c$.

- Let *ab.stim* represent the stimulation level of the ARB *ab*.
- Let *ab.resources* represent the number of resources held by the ARB *ab*.
- Let $TotalNumResources$ represent the total number of system wide resources allowed.

3.2.1. *Initialization*
The first stage of the algorithm, initialization, can primarily be thought of as a data pre-processing stage combined with a parameter discovery stage. During initialization, first all items in the data set are normalized such that the Euclidean distance[2] between the feature vectors of any two items is in the range of [0,1]. This can be performed through a variety of methods and could also be performed as a true pre-processing stage before the algorithm begins. It is important to note that, while for the current investigation Euclidean distance is the primary metric of both affinity and stimulation, other functions could be employed as well. What is important about this normalization is only that the range of possible reactions from cell-to-cell interaction remains within the range of [0,1]. After normalization, the affinity threshold is calculated. The affinity threshold is the average affinity value over all training data items' feature vectors. The affinity threshold is calculated as described in equation (1) below:

$$\text{affinity threshold} = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \text{affinity}(ag_i, ag_j)}{\frac{n(n-1)}{2}} \qquad (1)$$

where n is the number of training data items (antigens) in question, ag_i and ag_j are the ith and jth training antigen in the antigen training vector, and affinity(x,y) returns the Euclidean distance between the two antigens' feature vectors.

The final step in initialization is the seeding of the memory cells and initial ARB population. This is done by randomly choosing 0 or more antigens from the set of training vector to be added to the set of memory cells and to the set of ARBs.

3.2.2. *Memory Cell Identification and ARB Generation*
Once initialization is complete, training proceeds as a one-shot incremental algorithm. That is, each element of the training data is

[2] Euclidean distance was chosen as an intial starting place for this prototype as it has been used in many standard machine learning algorithms. One exploration of AIRS that is yet to be undertaken is an examination of different distance metrics for these various calculations.

presented to the AIRS learning algorithm exactly once. The first step of this stage of the algorithm is memory cell identification and ARB generation. Given a specific training antigen, ag, find the memory cell, mc_{match}, that has the following property:

$$mc_{match} = \arg max_{mc \in MC_{ag.c}} stimulation(ag, mc) \quad (2)$$

where $stimulation(x, y)$ is defined as in equation (3) below:

$$stimulation(x, y) = 1 - \text{affinity}(x, y) \quad (3)$$

If $MC_{ag.c} \equiv \oslash$, then $mc_{match} \leftarrow ag$ and $MC_{ag.c} \leftarrow MC_{ag.c} \cup ag$. That is, if the set of memory cells of the same classification as the antigen is empty, then add the antigen to the set of memory cells and denote this newly added memory cell as the match memory cell, mc_{match}. It should be noted here that while the stimulation function for the current work relies solely on Euclidean distance, this need not necessarily be the case.

Once mc_{match} has been identified, this memory cell is used to generate new ARBs to be placed into the population of (possibly) pre-existing ARBs (*i.e.,* those ARBs left in the system from exposure to previous antigens). This is done through the method shown in Figure 1, where the function $makeARB(x)$ returns an ARB with x as the antibody of this ARB and where $mutate(x, b)$ is defined in Figure 2. In Figure 2, the function $drandom()$ returns a random value in

$MU \leftarrow \oslash$
$MU \leftarrow MU \cup makeARB(mc_{match})$
$stim \leftarrow stimulation(ag, mc_{match})$
$NumClones \leftarrow hyper\_clonal\_rate * clonal\_rate * stim$
while ($\mid MU \mid < NumClones$)
do
  $mut \leftarrow false$
  $mc_{clone} \leftarrow mc_{match}$
  $mc_{clone} \leftarrow mutate(mc_{clone}, mut)$
  if($mut \equiv true$)
    $MU \leftarrow MU \cup makeARB(mc_{clone})$
done
$AB \leftarrow AB \cup MU$

Figure 1. Hyper-Mutation for ARB Generation

```
mutate(x, b)
{
  foreach(x.f_i in x.f)
  do
    change ← drandom()
    change_to ← drandom()
    if(change < mutation_rate)
      x.f_i ← change_to * normalization_value
      b ← true
  done
  if(b ≡ true)
    change ← drandom()
    change_to ← (lrandom() mod nc)
    if(change < mutation_rate)
      x.c ← change_to
  return x
}
```

Figure 2. Mutation Routine

the range [0,1] and ($lrandom()$ mod nc) returns a random value in the range {0,nc}.

3.2.3. *Competition for Resources and Development of a Candidate Memory Cell*

At this point a set of ARBs (AB) exists which includes mc_{match}, mutations from mc_{match}, and (possibly) remnant ARBs from responses to previously encountered antigens. Recall that the AIRS algorithm is a one-shot algorithm, so while the discussion has been divided into separate stages, only one antigen goes through this entire process at time (with the obvious exception being the initialization stage which takes place over the entire data set before training begins). The goal of the next portion of the algorithm is to develop a candidate memory cell which is most successful in correctly classifying a given antigen, ag. This is done primarily through three mechanisms. The first mechanism is through the competition for system wide resources. Following the methods first outlined by [30] and more fully realized by [29], resources are allocated to a given ARB based on its normalized stimulation value, which is used as an indication of its fitness as a recognizer of ag. The second mechanism is through the use of mutation for diversification

and shape-space exploration. The third mechanism is the use of an average stimulation threshold as a criterion for determining when to stop training on ag.

Similar to principles involved in genetic algorithms, the AIRS algorithm employs a concept of fitness for survival of individuals within the ARB population. Survival of a given ARB is determined in a two-fold, interrelated manner. First, each ARB in the population AB is presented with the antigen ag to determine the ARB's stimulation level. This stimulation is then normalized across the ARB population based on both the raw stimulation level and the class of the given ARB ($ab.c$). Based on this normalized stimulation value, each $ab \in AB$ is allocated a finite number of resources. If this allocation of resources would result in more resources being allocated across the population than allowed, then resources are removed from the weakest (least stimulated) ARBs until the total number of resources in the system returns to the number of resources allowed. Those ARBs which have zero resources are removed from the ARB population. This process is formalized in Figure 3.

While 3.3 will discuss this in more detail, two key aspects of this resource allocation routine for the initial formulation of the AIRS algorithm are noted here. First, the stimulation value of an ARB is not only determined by the stimulation function in equation 3 but is also based on the class of the ARB. The stimulation calculation method outlined in Figure 3 provides reinforcement both for those ARBs of the same class as ag that are highly stimulated by ag and for those ARBs that are of a different class from ag that do not exhibit a strong positive reaction to ag. Second, the distribution of resources is also based on the class of the ARB. This is done to provide additional reinforcement for those ARBs of the same class as ag without losing the potentially positive qualities of the remaining ARBs for reaction to future antigens.

At this point in the algorithm, the ARB population AB consists of only those ARBs that were most stimulated by the given antigen, ag, or more specifically, AB now consists of those ARBs that were able to successfully compete for resources. The algorithm continues first by determining if the ARBs in AB were stimulated enough by ag to stop training on this item. This is done by defining a vector $\vec{s}$ that is nc in length to contain the average stimulation value for each class subset of AB. That is:

$$s_i \leftarrow \frac{\sum_{j=1}^{|AB_i|} ab_j.stim}{\mid AB_i \mid}, ab_j \in AB_i$$

The stopping criterion is reached iff $s_i \geq stimulation\_threshold$ for all elements in $\vec{s} = \{s_1, s_2, \ldots s_{nc}\}$.

$minStim \leftarrow MAX$
$maxStim \leftarrow MIN$
foreach($ab \in AB$)
do
  $stim \leftarrow stimulation(ag, ab)$
  if ($stim < minStim$)
    $minStim \leftarrow stim$
  if ($stim > maxStim$)
    $maxStim \leftarrow stim$
  $ab.stim \leftarrow stim$
done
foreach($ab \in AB$)
do
  if($ab.c \equiv ag.c$)
    $ab.stim \leftarrow \frac{ab.stim - minStim}{maxStim - minStim}$
  else
    $ab.stim \leftarrow 1 - \frac{ab.stim - minStim}{maxStim - minStim}$
  $ab.resources \leftarrow ab.stim * clonal\_rate$
done
$i \leftarrow 1$
while($i \leq nc$)
do
  $resAlloc \leftarrow \sum_{j=1}^{|AB_i|} ab_j.resources, ab_j \in AB_i$
  if($i \equiv ag.c$)
    $NumResAllowed \leftarrow \frac{TotalNumResources}{2}$
  else
    $NumResAllowed \leftarrow \frac{TotalNumResources}{2*(nc-1)}$
  while($resAlloc > NumResAllowed$)
  do
    $NumResRemove \leftarrow resAlloc - NumResAllowed$
    $ab_{remove} \leftarrow argmin_{ab \in AB_i}(ab.stim)$
    if($ab_{remove}.resources \leq NumResRemove$)
      $AB_i \leftarrow AB_i - ab_{remove}$
      $resAlloc \leftarrow resAlloc - ab_{remove}.resources$
    else
      $ab_{remove}.resources \leftarrow ab_{remove}.resources - NumResRemove$
      $resAlloc \leftarrow resAlloc - NumResRemove$
  done
  $i \leftarrow i + 1$
done

Figure 3. Stimulation, Resource Allocation, and ARB Removal

Regardless of whether the stopping criterion is met or not, the algorithm proceeds by allowing each ARB in AB the opportunity to produce mutated offspring. While this adding of mutated offspring is similar to the method outlined in Figure 1, there are a few differences. This modified mutation generation routine is presented in Figure 4.

```
MU ← ∅
foreach(ab ∈ AB)
do
  rd ← drandom()
  if(ab.stim > rd)
    NumClones ← ab.stim * clonal_rate
    i ← 1
    while(i ≤ NumClones)
    do
      mut ← false
      ab_clone ← ab
      ab_clone ← mutate(ab_clone, mut)
      if(mut ≡ true)
        MU ← MU ∪ ab_clone
      i ← i + 1
    done
done
AB ← AB ∪ MU
```

Figure 4. Mutation of Surviving ARB

After allowing each surviving ARB the opportunity to produce mutated offspring, the stopping criterion is examined. If the stopping criterion is met, then training on this one antigen stops. If the stopping criterion has not been met, then this entire process, beginning with the method outlined in Figure 3, is repeated until the stopping criterion is met. The only exception to this repetition is that on every pass through this portion of the algorithm, except the first pass already discussed, if the stopping criterion is met after the stimulation and resource allocation phase, then the production of mutated offspring is not performed. Once the stopping criterion has been met, then the candidate memory cell is chosen. The candidate memory cell, $mc_{candidate}$, is the feature vector and class of the ARB that existed in the system before the most recent round of mutations that was the most stimulated ARB of the same class as the training antigen ag.

3.2.4. *Memory Cell Introduction*
The final stage in the training routine is the potential introduction of the just-developed candidate memory cell, $mc_{candidate}$, into the set of existing memory cells MC. It is during this stage that the affinity threshold calculated during initialization becomes critical as it dictates whether the $mc_{candidate}$ replaces mc_{match} that was previously identified. The candidate memory cell is added to the set of memory cells only if it is more stimulated by the training antigen, ag, than mc_{match}, where stimulation is defined as in equation (3). If this test is passed, then if the affinity between $mc_{candidate}$ and mc_{match} is less than the product of the affinity threshold and the affinity threshold scalar, then $mc_{candidate}$ replaces mc_{match} in the set of memory cells. This memory cell introduction method is presented in figure 5.

$CandStim \leftarrow stimulation(ag, mc_{candidate})$
$MatchStim \leftarrow stimulation(ag, mc_{match})$
$CellAff \leftarrow affinity(mc_{candidate}, mc_{match})$
if($CandStim > MatchStim$)
  if($CellAff < AT * ATS$)
    $MC \leftarrow MC - mc_{match}$
  $MC \leftarrow MC \cup mc_{candidate}$

Figure 5. Memory Cell Introduction

Once the candidate memory cell has been evaluated for addition into the set of established memory cells, training on this one antigen is complete. The next antigen in the training set is then selected, and the training process proceeds with memory cell identification and ARB generation. This process continues until all antigens have been presented to the system.

3.2.5. *Classification*
After training has completed, the evolved memory cells are available for use for classification. The classification is performed in a k-nearest neighbor approach. Each memory cell is iteratively presented with each data item for stimulation. The system's classification of a data item is determined by using a majority vote of the outputs of the k most stimulated memory cells.

3.3. Discussion of the AIRS Algorithm

While 3.2 presents a fairly formal overview of the AIRS algorithm, this section provides a conceptual discussion of some of the key elements of this classification system. The goal of the algorithm is the development of a set of memory cells that can be used to classify data. These artificial memory cells embody several characteristics seen in natural immune systems. Primarily, the memory cells are based on memory B Cells that have undergone a maturation process in the body. In mammalian immune systems, these memory B Cells are easily stimulated by invading antigens and undergo a process of hyper-somatic mutation as a response to recognized invading pathogens. The embodiment of this concept is seen in the function of memory cells in the AIRS algorithm. The artificial memory cells can also be said to take on the role of T Cells and Antigen Presenting Cells to some degree. In natural immune systems T Cells tend to be associated with a specific population of B Cells. When a T Cell recognizes an antigen, it then presents this antigen to the B Cells associated with it for further recognition. In the AIRS algorithm, this can be seen in the initial stages of identifying the matching memory cell which in turn develops a population of ARBs closely related to the matching memory cell.

The heart of the AIRS algorithm is the process of evolving memory cells from a population of ARBs. This evolutionary process has several key concepts which warrant mention here. The primary mechanism for providing evolutionary pressure to the population of ARBs in the development of memory cells is the competition for system wide resources. This concept, inspired by [29], is the means by which cell survival is determined and reinforcement for quality classification is provided. Like genetic algorithms, the goal of resource competition is the development of the fittest individuals. In the AIRS algorithm, fitness is initially determined by stimulation response of an individual ARB to an antigen. In the initial formulation of AIRS work, it is necessary not only to examine the stimulation response of a cell to an antigen but also to take into account the cell's class when compared to the antigen's class. For this reason, in the AIRS algorithm, cells with high stimulation responses that are of the same class as the antigen and cells with low stimulation responses that are not of the same class as the antigen are rewarded most heavily. The reward comes in the form of being allocated more system wide resources. On the other hand, those cells with low stimulation values but the same class or high stimulation values but a different class as the antigen are seen as not just poor classifier cells, but potentially detrimental classifier cells and they are thus rewarded less. Since those ARBs with the least ability to acquire resources are

purged from the system, there is great pressure to evolve toward a place in the search space that will provide the most reward. However, as we wish to maintain the generalizing capabilities of the system, the exact "amount" of error is not provided as feedback to an individual cell, rather those cells with higher quality representations are proided reinforcement.

ARBs which survive the competition for resources are further rewarded by being given the opportunity to produce mutated offspring. Again, as is the case with genetic algorithms, this competition for survival can also be seen in a truly evolutionary sense. That is, while the fittest individuals in a given round of antigen exposure might not survive to actually become a memory cell, their offspring might. Thus it is survival of a "species" of cell that this algorithm promotes. This is accomplished through the use of feature mutation in the training routine. The introduction of mutated offspring into the ARB population provides for a more thorough exploration of the search space. This exploration is further enhanced by the use of a stimulation threshold that must be met before the ARB population can be said to have "learned" to recognize a given antigen. This increases the evolutionary pressure to develop cells across the population which exhibit qualities desirable for memory cell inclusion.

After an ARB has successfully competed for resources among the general ARB population, if it was the most stimulated in response to the training antigen and was of the same class as the antigen, then it has the opportunity to be added to the pool of memory cells. When an antigen is first introduced to the system, the memory cell which is most stimulated by and of the same class as the antigen is allowed to inject mutated offspring into the ARB general population. However, at the end of the evolutionary process of this ARB population, this original memory cell can potentially be replaced by the evolved memory cell. This occurs only when the evolved memory cell is closer in the search space to the training antigen than the originally located memory cell and when these two memory cells are "close enough"(as determined by a user-parameter) to each other as well. From a machine learning point of view, this process is what provides for the data reduction capabilities of the AIRS algorithm. By allowing better classifying memory cells that occupy near-by locations in the shape space to replace existing memory cells, the AIRS algorithm reduces the number of cells needed to represent the problem domain. The algorithm also generalizes since the evolved memory cells in the system are not necessarily identical to any training instances.

3.4. RESULTS AND DISCUSSION

AIRS was tested on a number of benchmark data sets in order to assess the classification performance. This section will briefly highlight those results and discuss potential improvements for the algorithm; however, more details can be found in [32]. Once a set of memory cells has been developed, the resultant cells can be used for classification. This is done through a k-nearest neighbor approach. Experiments were undertaken using a simple linearly separable data set, where classification accuracy of 98% was achieved using a k-value of 3. This seemed to bode well, and further experiments were undertaken using the Fisher Iris data set, Pima diabetes data, Ionosphere data and the Sonar data set, all obtained from the repository at the University of California at Irvine [4]. Table II shows the performance of AIRS on these data sets when compared with other popular classifiers [12] and [13], and a discussion of these comparative results can be found in (Watkins and Boggess 2002a)[3].

These results were obtained from averaging multiple runs of AIRS, typically consisting of three or more runs and five-way, or greater, cross validation. More specifically, for the Iris data set a five-fold cross validation scheme was employed with each result representing an average of three runs across these five divisions. To remain comparable to other experiments reported in the literature, the division between training and test sets of the Ionosphere data set as detailed in [4] was maintained. However, the results reported here still represent an average of three runs. For the Diabetes data set a ten-fold cross validation scheme was used, again with each of the 10 testing sets being disjoint from the others, and results were averaged over three runs across these data sets. Finally, the Sonar data set utilized the thirteen-way cross validation suggested in the literature [4] and was averaged over ten runs to allow for more direct comparisons with other experiments reported in the literature. During the experimentation, it was noted by the authors that varying system parameters such as number of seed cells varied performance on certain data sets, however, varying system resources (i.e., the numbers of resources an ARB could compete for) seemed to have little affect. A comparison was made between the performance of AIRS and other benchmark techniques, where AIRS seemed not to outperform specialist techniques, but did outperform more general purpose algorithms, such as C4.5. To save duplication, the reader is referred to [31] for a detailed account of classification accuracy comparisons. Even though initial results from AIRS are promising, it can be said there are

[3] For the Diabetes data set, 11 others reported with lower scores, including Bayes, Kohonen, kNN, ID3...

Table II. Comparison of AIRS and Other Classifiers Classification Results on Benchmark Data

	Iris		Ionosphere		Diabetes		Sonar	
1	Grobian (rough)	100%	3-NN + simplex	98.7%	Logdisc	77.7%	TAP MFT Bayesian	92.3%
2	SSV	98.0%	3-NN	96.7%	IncNet	77.6%	Nave MFT Bayesian	90.4%
3	C-MLP2LN	98.0%	IB3	96.7%	DIPOL92	77.6%	SVM	90.4%
4	PVM 2 rules	98.0%	MLP + BP	96.0%	Linear Disc. Anala.	77.5-77.2%	Best 2-layer MLP + BP, 12 hidden	90.4%
5	PVM 1 rule	97.3%	**AIRS**	**94.9%**	SMART	76.8%	MLP+BP, 12 hidden	84.7%
6	**AIRS**	**96.7%**	C4.5	94.9%	GTO DT (5xCV)	76.8%	MLP+BP, 24 hidden	84.5%
7	FuNe-I	96.7%	RIAC	94.6%	ASI	76.6%	1-NN, Manhattan	84.2%
8	NEFCLASS	96.7%	SVM	93.2%	Fischer discr. anal.	76.5%	**AIRS**	**84.0%**
9	CART	96.0%	Non-linear perceptron	92.0%	MLP+BP	76.4%	MLP+BP, 6 hidden	83.5%
10	FUNN	95.7%	FSM + rotation	92.8%	LVQ	75.8%	FSM - method?	83.6%
11			1-NN	92.1%	LFC	75.8%	1-NN Euclidean	82.2%
12			DB-CART	91.3%	RBF	75.7%	DB-CART, 10xCV	81.8%
13			Linear perceptron	90.7%	NB	75.5-73.8%	CART, 10xCV	67.9%
14			OC1 DT	89.5%	kNN, k=22, Manh	75.5%		
15			CART	88.9%	MML	75.5%		
...					...			
22					**AIRS**	**74.1%**		
23					C4.5	73.0%		

a number of potential areas for simplification and improvement. There is clearly a need to understand exactly why and how AIRS behaves the way it does. This can be achieved through a rigorous analysis of the algorithm, and through emperical examination of the behavior of the ARB pool and memory set over time.

To date, the focus has been primarily on the classification performance of AIRS. [17] performs some of this empirical exploration by applying AIRS to a variety of classification problems in which the number of class ranged from 3 to 12 and the number of features ranged from 4 to 279. In the course of this work, AIRS was found to have the best performance of any single classifier on the publicly available credit.crx problem, of any known to the authors. Further emperical exploration of the AIRS algorithm is detialed in the description of two other suites of experiments: [18] discusses the effects of replacing the algorithm for evolving a candidate memory cell from the ARB pool and concludes that most of the effectiveness of the classifer lies in the replacement strategies for the memory cell pool itself. [24] performs limited exploration of modifications to the resource allocation mechanism as well as a more thorough examination of the tie-breaking mechanism in the k-nn algorithm. In the course of this latter experimentation, it was found that AIRS outperforms the best reported accuracy for the E.coli data set found in the UCI repository [4].

The majority of AIS techniques use the metaphor of somatic hypermutation or affinity proportional mutation. To date, AIRS does not employ this metaphor but instead uses a naive random generation of mutations. The remaining sections of this paper detail investigations into the behavior of the algorithm and present a modified version of AIRS, which is more efficient in terms of ARB production and employs affinity proportional mutation, and assess what, if any, difference these changes have made to the overall algorithm.

4. A More Efficient AIRS

This section details observations that have been made through a thorough investigation into AIRS and how issues raised through these observations have been overcome.

4.1. OBSERVATIONS

4.1.1. *The ARB Pool*

A crude visualization[4] was used to gain a better understanding of the development of the ARB pool. In AIRS there are two independent pools of cells, the memory cell pool and the ARB pool. The original formulation of AIRS uses the ARB pool to evolve a candidate memory cell of the same class as the training antigen, which can potentially enter the memory cell pool. During this evolution, ARBs of a different class than the training antigen were also maintained in the ARB pool. The stimulation of an ARB was based both on affinity to the antigen and on class, where highly stimulated ARBs were those of the same class as the antigen which were "close" to the antigen, or were of a different class and "far" from the antigen. However, the visualization, which consisted of an animated GIF file which was the concatenation of a series of plots of the ARB pool at each iteration in the pool's evolution on simple 2-dimensional simulated data, revealed that, during the process of evolving a candidate memory cell, evolving ARBs that are of a different class than the training antigen was wasted effort. The point of the interaction of the ARB pool with the antigenic material is really only in evolving a good potential memory cell, and this potential memory cell must be of the same class as the training antigen. The visualization demonstrates that there is a process of convergence by ARBs of the same class to the training antigen. Naturally, based on the reward scheme, ARBs of a different class are moving further away from the training antigen. However, this process essentially must start over for the introduction of each new antigen, and, therefore, previously existing ARBs are fairly irrelevant. Since there are two separate cell pools, with the true memory of the system only being maintained in the Memory Cell pool, maintaining any type of memory in the ARB pool made no effective contribution. Eliminating the maintenance of multiple classes in the ARB pool while generating a candidate memory cell simplifies the algorithm, reduces the memory required during execution, and improves the overall runtime.

4.1.2. *Mutation of Cells*

Motivated by observing the success of other AIS work, as well as by some of the tendencies discussed in [31] and [33], attention was paid to the way in which mutation occured within AIRS. In these two works, the authors notice that some of the evolved memory cells do not seem as high in quality as others. Additionally, it was observed that there seemed to be some redundancy in the memory cells that were produced. In [10] and other AIS work, mutation within an antibody or B-Cell is

[4] http://www.cs.kent.ac.uk/people/rpg/abw5/ARB_hundred.html

based on its affinity, so that high affinity cells undergo mutation with a more restricted range than lower affinity cells. These other AIS works have used this method of somatic hypermutation to a good degree of success. It was thought that embedding some of this approach in AIRS might result in higher quality, less redundant, memory cells. This approach was therefore adopted within AIRS.

4.2. AIRS: WHAT IS NEW?

For the remainder of this section changes that have been made to the AIRS algorithm are described. There then follow empirical results from the new formulation and a discussion of the implications of these results.

4.2.1. *Memory Cell Evolution*

In the original version of AIRS both ARBs "near" the antigen and of the same class as the antigen were rewarded and ARBS "far" from the antigen and of a different class than the antigen were rewarded. Also, ARBs were allowed to mutate their class values (mutate in this case means switching classes). In the newly revised version of AIRS, only ARBs of the same class are maintained in the ARB pool and mutation of the class value is no longer permitted. Figure 6 presents the chanes to the algorithm presented in Figure 3.

Recall that the stimulation threshold was originally used as a stopping criterion for training the ARB pool on an antigen. In order to stop training on an antigen the average normalized stimulation level had to exceed the stimulation threshold for each class group of ARBs. That is, in a 2-class problem, for example, the average normalized stimulation level of all class 0 ARBs had to be above the stimulation threshold, and the average normalized stimulation level of all class 1 ARBs had to be above the stimulation threshold. It was possible, and frequently the case in fact, that the average normalized stimulation level for the ARBs of the same class as the training antigen reached the stimulation threshold before the average normalized stimulation level of ARBs in different classes from the antigen. What this did, in effect, was allow for the evolution of even higher stimulated ARBs of the same class while they were waiting for the other classes to reach the stimulation threshold. By taking out these extra cycles of evolution which were du to ARBs of different classes, it is possible that the ARBs will not have converged "as much" as in the previous formulation. This can be overcome by raising the stimulation threshold and thus requiring a greater level of convergence.

$minStim \leftarrow MAX$
$maxStim \leftarrow MIN$
foreach($ab \in AB$)
do
  $stim \leftarrow stimulation(ag, ab)$
  if ($stim < minStim$)
    $minStim \leftarrow stim$
  if ($stim > maxStim$)
    $maxStim \leftarrow stim$
  $ab.stim \leftarrow stim$
done
foreach($ab \in AB$)
do
  $ab.stim \leftarrow \frac{ab.stim - minStim}{maxStim - minStim}$
  $ab.resources \leftarrow ab.stim * clonal\_rate$
done
$resAlloc \leftarrow \sum_{j=1}^{|AB_{ag.c}|} ab_j.resources, ab_j \in AB_{ag.c}$
$NumResAllowed \leftarrow TotalNumResources$
while($resAlloc > NumResAllowed$)
do
  $NumResRemove \leftarrow resAlloc - NumResAllowed$
  $ab_{remove} \leftarrow argmin_{ab \in AB_{ag.c}}(ab.stim)$
  if($ab_{remove}.resources \leq NumResRemove$)
    $AB_{ag.c} \leftarrow AB_{ag.c} - ab_{remove}$
    $resAlloc \leftarrow resAlloc - ab_{remove}.resources$
  else
    $ab_{remove}.resources \leftarrow ab_{remove}.resources - NumResRemove$
    $resAlloc \leftarrow resAlloc - NumResRemove$
done

Figure 6. Stimulation, Resource Allocation, and ARB Removal: Revised

4.2.2. *Somatic Hypermutation*
To explore the role of mutation on the quality of the memory cells evolved, the mutation routine was modified so that the amount of mutation allowed to a given gene in a given cell is dictated by the cell's stimulation value. Specifically, the higher the normalized stimulation value, the smaller the range of mutation allowed. Essentially, the range of mutation for a given gene = 1.0 - the normalized stimulation value of the cell. Mutation is then controlled over this range with the original

gene value being placed at the center of the range. This, in a sense, allows for tight exploration of the space around high quality cells, but allows lower quality cells more freedom to explore widely. In this way, both local refinement and diversification through exploration are achieved. This change is illustrated in figure 7, which is presented in a similar manner to figure 2.

```
mutate(x, b)
{
  range ← 1 − x.stim
  foreach(x.f_i in x.f)
  do
    change ← drandom()
    change_to ← drandom()
    bottom ← x.f_i / normalization_value − range/2
    if (bottom < 0)
      bottom ← 0
    change_to ← (change_to ∗ range) + bottom
    if(change_to > 1)
      change_to ← 1
    if(change < mutation_rate)
      x.f_i ← change_to ∗ normalization_value
      b ← true
  done
  return x
}
```

Figure 7. Mutation Routine: Revised

4.3. THE AIRS2 ALGORITHM

The changes made to the AIRS algorithm are small, but end up having an interesting impact on both the simplicity of implementation and on the quality of results. Section 5 will offer more discussion by way of comparison. For now, the changes to the original AIRS presented in section 3 will be discussed. These can be identified as follows:

– Only the Memory Cell pool is seeded during initialization rather than both the **MC** pool **(M)** and the ARB pool **(P)**. Since we are no longer concerned about maintaining memory or class diversity

within P it is no longer necessary to initialize P from the training data or from examples of multiple classes.

- During the clonal expansion from the matching memory cell used to populate **P**, the newly created ARBs are no longer allowed to mutate class. Again, maintaining class diversity in **P** is not necessary.

- Resources are only allocated to ARBs of the same class as the antigen and are allocated in proportion to the ARB's stimulation level in reaction to the antigen.

- During affinity maturation (mutation), a cell's stimulation level is taken into account. Each individual gene is only allowed to change over a finite range. This range is centered at the gene's pre-mutation value and has a width the size of the difference of 1.0 and the cell's stimulation value. In this way the mutated offspring of highly stimulated cells (those whose stimulation value is closer to 1.0) are only allowed to explore a very tight neighborhood around the original cell, while less stimulated cells are allowed a wider range of exploration. (It should be noted that during initialization all gene values are normalized so that the Euclidean distance between any two cells is always within one. During this normalization, the values to transform a given gene to within the range of 0 and 1 are discovered, as well. This allows for this new mutation routine to take place in a normalized space where each gene is in the range of 0 and 1.)

- The training stopping criterion no longer takes into account the stimulation value of ARBs in all classes, but now only accounts for the stimulation value of the ARBs of the same class as the antigen.

4.4. RESULTS AND DISCUSSION

To allow for comparison between the two versions of the algorithm, the same experiments that were performed on the original AIRS were performed on the new formulation of AIRS (AIRS2). Section 5 will provide a more thorough comparative discussion, but for now, results of AIRS2 on the four previously discussed benchmark sets are presented in table III.

These results were obtained by following the same methodology as the original results reported in section 3.4, which is elaborated upon in [31] and [32]. Again, we note that these results are competitive with

Table III. AIRS2 Classification Results on Benchmark Data

Iris	Ionosphere	Diabetes	Sonar
96.0%	95.6%	74.2%	84.9%

other classification techniques discussed in the literature, such as C4.5, CART, and Multi-Layer Perceptrons (as presented in table II).

5. Comparative Analysis

This section briefly touches on some comparisons between the original version of AIRS presented in discussed in section 3 (AIRS1) and the revisions to this algorithm presented in section 4. The focus of this discussion will be on two of the more important features of the AIRS algorithms: classification accuracy and data reduction. When undertaking revisions and refinements of an algorithm, it is necessary to check also that such changes have not unduly affected how the user defined parameters are effected. To this end, this section also presents analysis of altering various user defined parameters, specifically number of seed cells and mutation rate and how they effect the classification accuracy. This seems to be rarely done in the literature, but a detailed analysis of parameter adjustment for an immune network model can be found in [28].

5.1. CLASSIFICATION ACCURACY

The success of AIRS1 as a classifier (cf, [32]) makes it important to assess any potential changes to the algorithm in light of test set classification accuracy. To aid in this task, Table IV presents the best average test set accuracies, along with the standard deviations, achieved by both versions of AIRS on the four benchmark data sets. All experiments were repeated in the same way, using the same parameters as the original work.

It can be noted that the revisions to AIRS presented in section 3 do not require a sacrifice in classification performance of the system. In fact, for 3 of the 4 data sets we see a slight improvement in the accuracy; however, these differences are not statistically significant. What is important to note is that the changes introduce no fundamental differences in classification accuracy of the system.

Table IV. Comparative Average Test Set Accuracies

	AIRS1: Accuracy	AIRS2: Accuracy
Iris	96.7 (3.1)	96.0 (1.9)
Ionosphere	94.9 (0.8)	95.6 (1.7)
Diabetes	74.1 (4.4)	74.2 (4.4)
Sonar	84.0 (9.6)	84.9 (9.1)

5.1.1. *Effect of Parameters on Classification Accuracy*
The previous section illustrates that the classification accuracy of AIRS has not diminished It is also important to establish that the behavior of AIRS has not been altered with respect to the user defined parameters. In order to establish this, investigations were undertaken to determine what affect altering the number of seed cells might have on classification accuracy (figures 8 and 9) and how altering the mutation rate also might affect the classification accuracy when compared over both systems (figures 10 and 11).

Figure 8 shows how altering the number of initial seed cells in AIRS affects the overall classification performance for AIRS1. Here it can be observed that, on average across the four data sets, increasing the number of seed cells there aphas little impact on accuracy. For AIRS1 an exception to this overall trend is the Sonar data set, which benefits from increasing the number of seed cells.

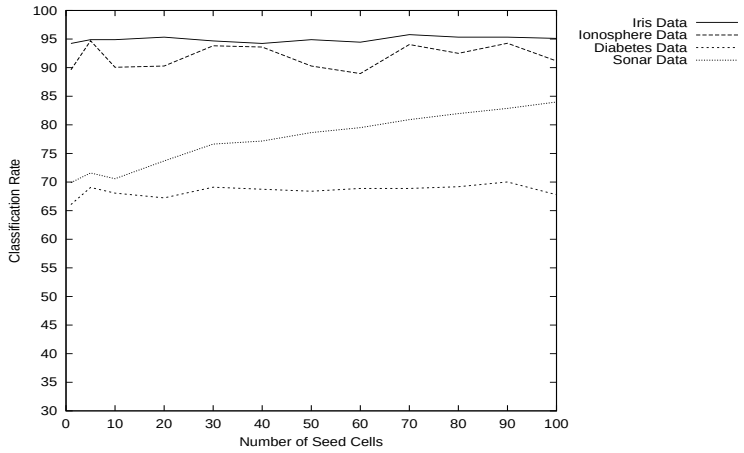

Figure 8. The affect of altering the number of seed cells on classification accuracy - AIRS Version 1

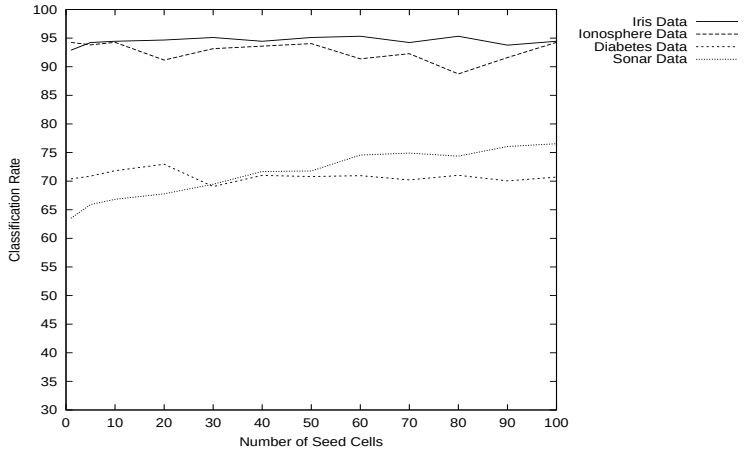

Figure 9. The effect of altering the number of seed cells on classification accuracy - AIRS Version 2

As shown in figure 9, AIRS2 performs comprably to AIRS 1 with respect to increasing the number of seed cells, with the exception that the Sonar problem no longe shows improvement from increasing the number of seed cells.

Another key user defined parameter is the mutation rate. This affects how many mutations are performed when a new ARB is created—clearly the higher the mutation rate, the wider the search space covered. Figures 10 and 11 show the effects on classification accuracy of altering the mutation rate with respect to classification accuracy. In figure 10, results for the sonar data and some of the diabetes are not presented due to the fact that as the mutation rate increases, more potential candidate cells are produced increasing the memory usage and computational complexity. The reader is directed to [1] and [9] for more discussion of the effect. [18] present alternatives to candidate cell production in AIRS2 which are not subject to this effect and yet produce comparably good classification results.

5.2. DATA REDUCTION

From the previous subsection it can be seen that the changes introduced to AIRS offer no real difference in classification accuracy, so the question arises: why bother? Why introduce these changes to a perfectly reasonably performing classification algorithm? The answer lies in the data reduction capabilities of AIRS. In [31] and [33], the authors discuss that aside from competitive accuracies another intriguing feature of the AIRS classification system is its ability to reduce

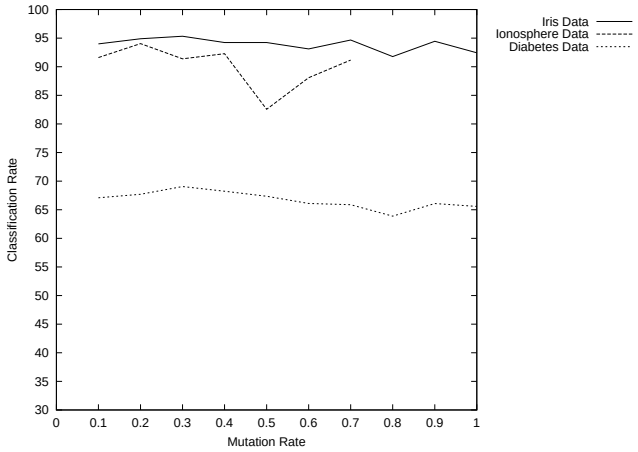

Figure 10. The affect of altering the mutation rate on classification performance - AIRS Version 1

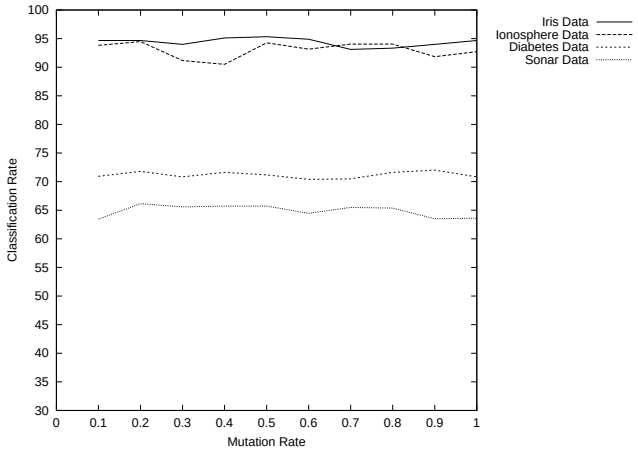

Figure 11. The affect of altering the mutation rate on classification performance - AIRS Version 2

the number of data points needed to characterize a given class of data from the original training data to the evolved set of memory cells. Given the volumes of data associated with many real-world data sets of interest, any technique that can reduce this volume while retaining the salient features of the data set is useful. Additionally, it is this collection of memory cells that are the primary classifying agents in the evolved system. Since classification is currently performed in a k-nearest neighbor approach, for which classification time is dependent upon the number of data points, any reduction in the overall number

Table V. Comparison of the Average Size of the Evolved Memory Cell Pool

Training Set	Size	AIRS1: Memory Cells	AIRS2: Memory Cells
Iris	120	42.1/65% (3.0)	30.9/74% (4.1)
Ionosphere	200	140.7/30% (8.1)	96.3/52% (5.5)
Diabetes	691	470.4/32% (9.1)	273.4/60% (20.0)
Sonar	192	144.6/25% (3.7)	177.7/7% (4.5)

of evolved memory cells is useful. Table 5 presents the average size of the evolved set of memory cells and the amount of data reduction this represents in terms of population size and percentage reduction, along with standard deviations, for each version of the algorithm on the four benchmark data sets. The original training set size is also presented for comparison. There are two points of interest: 1) Both versions of the algorithm exhibit data reduction, and 2) AIRS2 tends to exhibit greater data reduction than AIRS1. This second point is the more important for our current discussion. As mentioned in section 4.3, one of the goals of the revision of the AIRS algorithm was to see if employing somatic hypermutation through a method more in keeping with other research in the AIS field would increase the efficiency of the algorithm. The current measure of efficiency under concern is the amount of data needed to represent the original training set to achieve accurate classifications. We can see from Table V that, for the majority of the data sets, AIRS2 was able to achieve accuracy comparable with AIRS, with greater efficiency. In fact for half of the data sets, Ionosphere and Diabetes, the degree of data reduction is greatly increased (from 30% to 52% for Ionosphere data and from 32% to 60% for the diabetes data set). In general, it appears that the revisions to AIRS provide greater data reduction, and hence greater efficiency, without sacrificing accuracy.

5.3. A WORD ABOUT SIMPLICITY

While the focus has not been on algorithmic complexity analysis of the two versions of AIRS for this current paper, it would be remiss not to make a brief mention concerning the simplifying effects of the revision to AIRS. As mentioned in section 4, the reformulation of AIRS was chiefly motivated by some basic observations about the workings of the system. One observation was that the original version of AIRS maintained representation of too many cells for its required task. This

led to the elimination of maintaining multiple classes of cells in the ARB pool or of retaining cells in the ARB pool at all. This has the simplifying effect of reducing the memory necessary to run the system successfully. A second observation concerning the quality of the evolved memory cells led to the investigation of the mutation mechanisms employed in the original algorithm. By adopting an approach to mutation proven to be successful in other AIS, it has been possible to increase the quality of the evolved memory cells that is evidenced by the increased data reduction without a decrease in classification accuracy. Both of these overarching changes (ARB pool representation and the mutation mechanisms used) have exhibited a simplifying effect on the classification system as a whole.

6. Conclusions and Future Work

This paper has focused on a supervised learning system based on immunological principles. The Artificial Immune Recognition System (AIRS) introduced in [31] exhibited initial success as a classification algorithm. However, as with any initial system, there were some revisions and refinements that could be made to AIRS that would decrease the complexity of the system. This paper has presented investigations for two of these revisions and it has been noted that these changes have not affected the overall classification accuracy of the system and have improved efficiency.

Future work lies in the application of AIRS to an immunological problem: predicting the binding or non-binding of T-cell receptors. This is a very large and complex problem domain, dealing with dimensions in excess of 2500. Initial experiments have shown that AIRS is more than comparable to traditional techniques such as neural networks which have been applied to this data, achieving similar classification accuracy, but with faster production of the classification model. This work will also be extended to a seven class T-cell binding prediction, as with all things in immunology, the two class binding problem is only the first stage in a very complex classification process.

References

1. Ayara, M., J. Timmis, R. D. Lemos, L. N. D. Castro, and R. Duncan: 2002, 'Negative Selection: How to Generate Detectors'. In: J. Timmis and P. Bentley (eds.): *Proceedings of 1st International Conference on Artificial Immune Systems (ICARIS)*. Canterbury, UK, pp. 89–98.

2. Bersini, H. and F. Varela: 1990, 'Hints for adaptive problem solving gleaned from immune networks'. In: *Parallel Problem Solving from Nature and 1st Workshop PPSW 1*. Dortmund and Federal Republic of Germany, pp. 343–354.
3. Bishop, C. M.: 1995, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press.
4. Blake, C. L. and C. J. Merz: 1998, *UCI Repository of Machine Learning Databases*. http://www.ics.uci.edu/ mlearn/MLRepository.html.
5. Bonabeau, E., M. Dorigo, and G. Theraulaz: 2000, *Swarm Intelligence: From Natural to Artifical Systems*. MIT Press.
6. Burnet, F.: 1959, *The clonal selection theory of acquired immunity*. Cambridge University Press.
7. Carter, J. H.: 2000, 'The immune systems as a model for pattern recognition and classification'. *Journal of the American Medical Informatics Association* **7**(1).
8. de Castro, L. N. and J. Timmis: 2002a, 'An artificial immune network for multimodal optimisation'. In: *Congress of Evolutionary Computation. Part of the World Congress on Computational Intelligence*. Honolulu, HI., pp. 699–704.
9. de Castro, L. N. and J. Timmis: 2002b, *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag.
10. de Castro, L. N. and F. von Zuben: 2000a, 'The clonal selection algorothm with engineering applications'. In: *Proceedings of Genetic and Evolutionary Computation*. Las Vegas, USA., pp. 36–37.
11. de Castro, L. N. and F. von Zuben: 2000b, 'An evolutionary immune network for data clustering'. In: *SBRN*. Brazil, pp. 187–204.
12. Duch, W.: 2000a, 'Datasets used for classification: Comparison of results'. http://www.phys.uni.torun.pl/kmk/projects/datasets.html.
13. Duch, W.: 2000b, 'Logical rules extracted from data'. http://www.phys.uni.torun.pl/kmk/projects/rules.html.
14. Farmer, J., N. Packard, and A. Perelson: 1986, 'The immune system and adaptation and machine learning'. *Physica D* **22**, 187–204.
15. Farmer, J. D.: 1990, 'A Rosetta Stone for Connectionism'. *Physica D* **42**, 153–187.
16. Forrest, S., A. Perleson, L. Allen, and R. Cherukuri: 1994, 'Self-Nonself discrimination in a computer'. In: *Proceedings of IEEE Symposium on Research in Security and Privacy*. Oakland, USA, pp. 202–212.
17. Goodman, D., L. Boggess, and A. Watkins: 2002, 'Artificial Immune System Classification of Multiple-Class Problems'. In: C. H. Dagli, A. L. Buczak, J. Ghosh, M. J. Embrechts, O. Ersoy, and S. W. Kercel (eds.): *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol. 12. New York, pp. 179–184.
18. Goodman, D., L. Boggess, and A. Watkins: 2003, 'An Investigation into the source of power for AIRS, an artificial Immune Classification System'. In: *Proceedings of the 2003 International Joint Conference on Neural Networks*.
19. Haykin, S.: 1999, *Neural Networks: A comprehensive foundation*. Upper Saddle River, NJ, USA: Prentice Hall.
20. Hofmeyr, S. and S. Forrest: 2000, 'Arichitecture for an Aritifcial Immune System'. *Evolutionary Computation* **7(1)**, 45–68.
21. Jerne, N.: 1974, 'Towards a network theory of the immune system'. *Annals of Immunology (Inst.Pasteur)* **125C**, 373–389.
22. Kennedy, J. and R. Eberhart: 2001, *Swarm Intelligence*. Morgan Kaufmann.

23. Kim, J. and P. Bentley: 2001, 'Towards an artificial immune system for network intrusion detection: An investigation of clonal selection with negative seelction'. In: *Proceedings of the Congress on Evolutionary Computation*. pp. 1244–1252.
24. Marwah, G. and L. Boggess: 2002, 'Artificial Immune Systems for Classification: Some Issues'. In: J. Timmis and P. J. Bentley (eds.): *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, Vol. 1. University of Kent at Canterbury, pp. 149–153.
25. Mitchell, M.: 1996, *An Introduction to Genetic Algorithms*. MIT Press.
26. Mitchell, T.: 1997, *Machine Learning*. McGraw-Hill.
27. Nasaroui, O. F., F. Gonzalez, and D. Dasgupta: 2002, 'The fuzzy artificial immune system: Motivations, basic concepts and application to clustering and web profiling'. In: *International Joint Conference on Fuzzy Systems. Part of the World Congress on Computational Intelligence*. Honolulu, HI., pp. 711–717.
28. Timmis, J.: 2000, 'Artificial immune systems: A novel data analysis technique inspired by the immune network theory.'. Ph.D. thesis, Department of Computer Science. University of Wales, Aberystwyth.
29. Timmis, J. and M. Neal: 2001, 'A Resource Limited Artificial Immune System'. *Knowledge Based Systems* **14**(3/4), 121–130.
30. Timmis, J., M. Neal, and J. Hunt: 2000, 'An Artificial Immune System for Data Analysis'. *Biosystems* **55**(1/3), 143–150.
31. Watkins, A.: 2001, 'A Resource Limited Artificial Immune Classifier'. Master's thesis, Mississippi Sate University.
32. Watkins, A. and L. Boggess: 2002a, 'A new classifier based on resource limited artificial immune systems'. In: *Congress on Evolutionary Computation. Part of the World Congress on Computational Intelligence*. Honolulu, HI., pp. 1546–1551.
33. Watkins, A. and L. Boggess: 2002b, 'A resource limited artificial immune classifier'. In: *Congress on Evolutionary Computation. Part of the World Congress on Computational Intelligence*. Honolulu, HI., pp. 926–931.
34. Watkins, A. and J. Timmis: 2002, 'Artificial Immune Recognition System (AIRS): Revisions and Refinements'. In: J. Timmis and P. J. Bentley (eds.): *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*. University of Kent at Canterbury, pp. 173–181.