ARTIFICIAL IMMUNE RECOGNITION SYSTEM (AIRS) A REVIEW AND ANALYSIS

Technical Report
No. 1-02

Jason Brownlee
Master of Information Technology, Swinburne University of Technology, 2004
Bachelor of Applied Science, Computing, Swinburne University of Technology, 2002
(jbrownlee@ict.swin.edu.au)

Centre for Intelligent Systems and Complex Processes (CISCP)
Faculty of Information & Communication Technologies (ICT)
Swinburne University of Technology (SUT)

January 2005

Abstract

The natural immune system is a robust and powerful information process system that demonstrates features such as distributed control, parallel processing and adaptation or learning via experience. Artificial Immune Systems (AIS) are machine-learning algorithms that embody some of the principles and attempt to take advantages of the benefits of natural immune systems for use in tackling complex problem domains. The Artificial Immune Recognition System (AIRS), is one such supervised learning AIS that has shown significant success on broad range of classification problems. The focus of this work is the AIRS algorithm, specifically the techniques history, previous research and algorithm function. Competence with the AIRS algorithm is demonstrated in terms of theory and application. The AIRS algorithm is analysed from the perspective of reasonable design goals for an immune inspired AIS and a number of limitations and areas for improvement are identified. A number of original and borrowed augmentations, simplifications and changes to the AIRS algorithm are then proposed to addresses the identified areas. A professional-level implementation of the AIRS algorithm is produced and is provided as a plug-in for the WEKA machine-learning workbench.

Acknowledgements

Firstly, I would that to sincerely thank my advisor Professor Tim Hendtlass and the CISCP for giving me the opportunity and financial support to research and work in a field that is both interesting and practical, and something I am passionate about. Secondly, thanks to the support provided by all the people at CISCP, for listening to my rants and for consistently providing positive and constructive feedback. Thanks to Jon Timmis and Donnald Goodman for providing soft copies of some hard-to-find conference and journal papers. Finally, thanks to Andrew Watkins for his support answering my most basic questions regarding his AIRS algorithm, and for providing papers and sample source code for study.

Table of Contents

List of Figures

List of Equations

List of Tables

1. Introduction

Artificial immune systems are a technique new to the scene of biological inspired computation and artificial intelligence, based on metaphor and abstraction from theoretical and empirical knowledge of the mammalian immune system. A robust biological process critical to the combating of disease in the body, the immune system is known to be distributed in terms of control, parallel in terms of operation, and adaptive in terms of function, all of which are features desirable for solving complex or intractable problems faced in the field of artificial intelligence. This document describes and reviews one implementation of artificial immune systems called AIRS (Artificial Immune Recognition System) designed for use with supervised learning (classification) problems.

Section 2. provides an overview of the main AIRS algorithm, focusing on the algorithms characteristics and features, and introduces some of the terminology from the field of AIS to explain the basic abstraction from the biological immune system that AIRS implements. Section 3. provides a more detailed review of the algorithm, explaining elements of the algorithms function and implementation. A detailed summary of the user-defined parameters used to tune the function of the algorithm is also provided, with a focus of known effects and common algorithm configuration. Section 4. reviews the history of the AIRS algorithm. This is discussed first from the perspective of precursor algorithms from which AIRS borrows or uses similar ideas or algorithm functionality. Next, a review of the research and evolution of the AIRS algorithm is provided, highlighting interesting findings and extensions to the canonical AIRS algorithm.

Finally, section 5. proposes extensions to the technique. The section starts with a review of desirable goals for an AIRS-like immune-inspired supervised learning system. Some of the features of AIRS are reiterated and a number of limitations and areas for improvement are identified. A set of AIRS-specific statistics are discussed for both the training of an AIRS classifier, and the resulting classifier itself. The section concludes with the proposal of a number of original and borrowed algorithm simplifications and extension that address all of the limitations and areas for improvement raised. The culmination is a revised AIRS or new AIRS-like algorithm whose performance is speculated to be similar to that of AIRS.

2. Overview of AIRS

An artificial immune system (AIS) is a class of adaptive or learning computer algorithm inspired by function of the biological immune system, designed for and applied to difficult problems such as intrusion detection, data clustering, classification and search problems. It is critical at the outset to stress that although terminology and function of AIS are described using biological terms from the field of immunological research, they are taken as simplifications and abstractions and not intended to be models or representative of immunological response systems.

The field of AIS research has been around for approximately 15 years, and for the majority of its history has been concerned with feature extraction (data clustering), and change or anomaly identification, such as intrusion detection systems. More recently, AIS

has been applied to broader domains such as function optimisation, and in the case of AIRS, classification. The recent shift in applicable problem domains has required a rethink of the algorithms application and adaptation of existing tried and tested AIS elements. This new field of research provides an opportunity for innovation, not only in the designing new specialised AIS algorithms, but also in the successful encoding the adaptive power of the metaphor.

2.1 Algorithm Characteristics

The AIRS algorithm was one of the first AIS technique designed specifically and applied to classification problems. It has been shown in [1] to exhibit the following desirable algorithmic characteristics:

Self-regulation – A problem common to the field of artificial neural networks is the selection of an appropriate topology or neuronal architecture. AIRS does not require the user to select an architecture, instead the adaptive process discovers or learns an appropriate architecture during training.

Performance – Empirical evaluation of the technique in [2] on standard classification problems from the University of California, Irvine [3], when compared to the empirical results of the best known classifiers from [4,5] show that AIRS is a competitive classification system. Results indicate that AIRS can achieve classification accuracy in the top five to top eight when ranked against some of the widely known best classification systems, and in the case of [6], is capable of achieving the best classification result known for some datasets.

Generalisation – Unlike techniques such as k-Nearest Neighbour that use the entire training dataset for classification, AIRS performs generalisation via data reduction. This means that the resulting classifier produced by the algorithm represents the training data with a reduced or minimum number of exemplars. It is typical for AIRS to produce classifiers with half the number of training instances [7].

Parameter Stability – The algorithm has a number of parameters that allows tuning of the technique to a specific problem, with the intent of achieving improved results. A feature of the algorithm is that over a wide range of parameter values, the technique is capable of achieving results within a few classification accuracy percentage points of the results achieved with an optimal parameter set.

2.2 AIRS and Terminology

The AIRS algorithm relies on a number of core principles from AIS research, all of which are loose abstractions from mammalian immunological research. Provided in this section is a description of some of the terminology used in the field of AIS to describe the AIRS algorithm. Using an abstraction of immune function, a simplified overview of the principle elements of AIRS is described. This section is by no means a complete review of immune system function or AIS background. For a in-depth review of AIRS related AIS terminology see [7]. For an overview of key AIS terms please see section 10. For a

review of AIS, see [8]. For a quick overview of immune function as it pertains to classification see [9].

A simplistic view of the immune system is that of an organ whose job it is to detect pathogens (potentially harmful material or **antigens**), and respond by protecting the organism from that material. The system is adaptive in that it improves in terms of antigen recognition over time. As more antigens are observed of similar characteristic, the more effective the system becomes at recognising and thus responding to that antigen. Response to an antigen comes in the form of an antibody whose job it is to neutralise the pathogenic material. In this abstraction of immune function, the elements that perform anomaly detection are referred to as **B-cells** and **T-cells**. The cells are suited to specific antigens and perform recognition or matching in **shape-space**, which is nothing more than the features of attributes of the antigen.

The term used to describe the degree of similarity between a recognition cell and an antigen is called **affinity**. The adaptive ability of the immune system is a process called **affinity maturation**. During an immune response the recognition cell will perform **clonal expansion,** which means it will generate many clones of itself in an attempt to gain a better match next time the antigen is seen. A process called **somatic hypermutation** mutates the generated clones in proportion to the affinity between the recognition cell and the antigen.

The clones produced have different receptors (features) to their parent, some of which are likely to be a better match to the antigen observed. Darwinian-like competition and selection between the resulting clones then occurs where only those cells with highest affinity with the antigen are maintained. This process is called **clonal selection**. The immune system is said to have a form of memory, in that through its interaction with antigens in the past, it is capable of remembering what a pathogen "looks like" and can better defend the organism in the future.

3. AIRS in Detail

The specification of the AIRS algorithm is reasonably complex at the implementation level. For a precise specification of the algorithm in pseudo code see [7]. This section provides an overview of the algorithm, focusing the discrete functional components of the technique. The review is intended to provide insight into the functional workings of the technique without getting too deep into implementation specific issues. Also provided in this section is a review of the configurable user parameters of the system and their impact on the functioning of the system. For a description of the user configurable parameters with an empirical view of their effect on the system, see [2].

It should be noted that this overview is restricted to the AIRS2 or version two of the AIRS algorithm [10]. It has been explained to the author (private communication) that the AIRS1 or version one of the technique [1,11] is now deprecated in favour of the second revision of the technique. Please see section 4. for a review of the AIRS1 and AIRS2 algorithms and their specific differences.

3.1 The Algorithm

The function of the AIRS algorithm is to prepare a pool of recognition or memory cells (data exemplars) which are representative of the training data the model is exposed to, and is suitable for classifying unseen data. The lifecycle of the AIRS system is as follows:

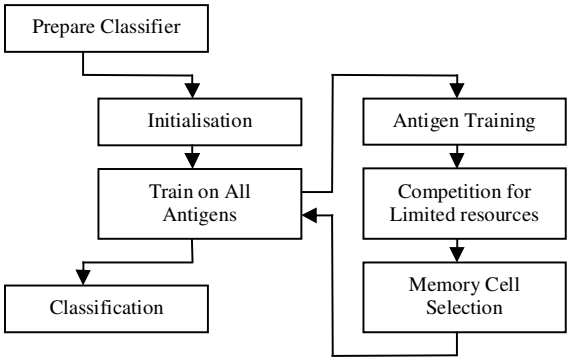

Figure 1 - Lifecycle overview of the AIRS algorithm

3.1.1 Initialisation

This step of the algorithm consists of preparing the data for use in the training process, and preparing system variables. The training data is normalised so that the range of each numeric attribute is in the range [0,1]. An affinity measure is required for use through the training process. The typical measure used is the inverted Euclidean distance. The important point here though is that the maximum distance measured between any two recognition cells or antigen and recognition cell (both simply data vectors) must also be in the range [0,1]. This can be achieved by adding the following step to the data normalisation process:

$$normalisedValue = normalisedValue \cdot \sqrt{\frac{1}{n}}$$

Equation 1 - Calculation for ensuring that the distance between any two vectors is between 0 and 1

where the normalised value is data attribute in the range of [0,1], and n is the number of attributes used in the distance evaluation. Another approach to ensuring the resulting distance values are in the range of [0,1], without requiring the data to be normalised is to simply divide the calculated Euclidean distance by the maximum distance between any two vectors. The following shows Euclidean distance, where *v1* and *v2* represent two elements that affinity is measured between and *n* is the number of attributes.

$$dist = \sqrt{\sum_{i=1}^{n} (v1_i - v2_i)^2}$$

Equation 2 - Euclidean distance measure

The maximum distance between any two data vectors is simply the root of the sum of the square ranges, where *r* is the known data range for attribute *i*.

$$\max Dist = \sqrt{\sum_{i=1}^{n} r_i^2}$$

Equation 3 - Maximum distance calculation

Affinity is a similarity value, this means that the smaller the affinity value the higher the affinity is said to be (closer the vectors are to each other).

$$affinity = (\frac{dist}{\max Dist})$$

Equation 4 - Affinity measure

It is important to note that Euclidean distance works well for numerical attributes but breaks down for nominal attributes especially common in UCI datasets. This can be overcome by assuming that difference between nominal attributes is binary (match or no match) and the attribute range is one.

The next step is to seed the memory cell pool. The memory cell pool is the collection of recognition elements that make up the classifier produced at the end of the training scheme. Seeding the memory pool is an optional step and involves randomly selecting a number of antigens to become memory cells

The final step during the initialisation is to prepare the *affinity threshold* (AT) system variable. The affinity threshold is the mean affinity between antigens in the training dataset. Either it can be calculated from a sample of the training set or the entire training set. This calculated value is then used later during the training scheme to determine whether candidate memory cells that are prepared, can replace existing memory cells in the classifier (explained later).

3.1.2 Antigen Training

The AIRS algorithm is a single-shot algorithm in that only one pass over the training data is required to prepare a classifier. Each antigen is exposed to the memory pool one at a time. The recognition cells in the memory pool are stimulated by the antigen and each cell is allocated a stimulation value (inverted affinity). The memory cell with the greatest stimulation is then selected as the best match memory cell for use in the affinity maturation process.

$$stim = 1 - affinity$$

Equation 5 - Stimulation equation

A number of mutated clones are then created from the selected memory cell and added to the ARB pool. An ARB (Artificial Recognition Ball) is an abstract concept and represents a number similar or identical recognition cells. The ARB pool is a work area where the AIRS system refines mutated clones of the best match memory cell for a specific antigen. The number of mutated clones created of the best match is calculated as follows:

$$numClones = stim \cdot clonalRate \cdot hypermutationRate$$

Equation 6 - The number of clones created for the best matching memory cell

where, the *stim* is the stimulation between the best match memory cell and the antigen. Both the *clonal rate* and the *hypermutation rate* are user-defined parameters.

3.1.3 Competition for Limited Resources

After a number of mutated clones of the best matching memory cell are added to the ARB pool, the process of ARB generation and competition begins. This process can be described in the following figure.

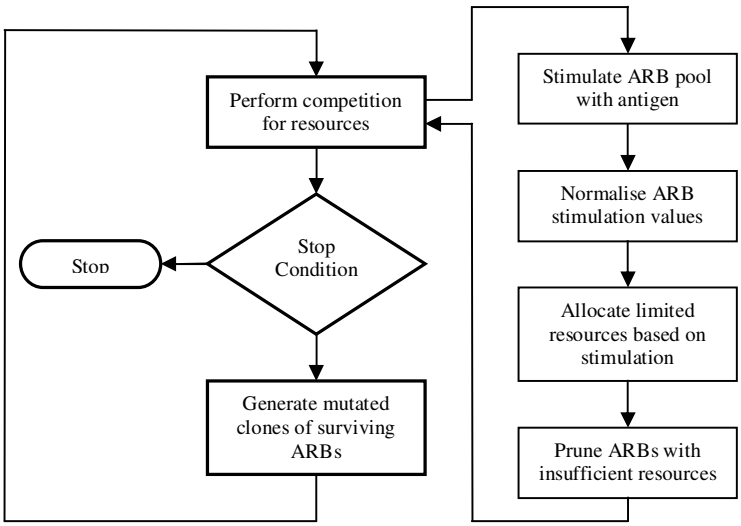

Figure 2 - ARB cell refinement through competition for limited resources

The process is quite simple from a high-level. Competition for limited resources is used to control the size of the ARB pool and promote those ARBs with greater stimulation (and thus affinity) to the antigen being trained on. The stop condition in the middle of the loop allows the final step of clone generation to be avoided when the ARB pool reaches a desirable state. In this process only ARBs of the same class as the antigen are considered, meaning that the class of an ARB is never adjusted in the mutation process. The final step sees each ARB in the pool have mutated clones generated using the same clonal

expansion and somatic hypermutation steps used previous to generate mutated clones of the best match from the memory cell. Here the number of clones generated for each ARB in the pool is calculated as the following:

$$numClones = stim \cdot clonalRate$$

Equation 7 - The number of clones created for an ARB in the ARB pool

In the resource allocation process, the amount of resources allocated to each ARB is as follows:

$$resource = normStim \cdot clonalRate$$

Equation 8 - Equation used for resource allocation

A user defined parameter *total resources* is defined that specified that maximum number of resources that can be allocated. During the resource allocation process the total resources allocated is determined and compared against the maximum total resources. The ARB pool is then sorted by allocated resources (decending) and resources are removed from ARBs starting at the end of the list until the total allocated resources is below the total resources allowed. Finally, those ARBs with zero resources are removed from the pool. The stop condition for this process of ARB refinement occurs when the mean normalised stimulation is more than the user defined *stimulation threshold*.

3.1.4 Memory Cell Selection

Once the stop condition for the ARB refinement process is completed, the ARB with the greatest normalised stimulation scoring is selected to become the memory cell candidate. The ARB is copied into the memory cell pool if the stimulation value for the candidate is better than that of the original best matching memory cell. A check is made to determine if the original best matching memory cell should be removed. This occurs if the affinity between the candidate memory cell and the best matching cell is less than a cut-off. This memory cell replacement cut-off is defined as:

$$cutOff = affinityThreshold \cdot affinityThresholdScalar$$

Equation 9 - Cut-off affinity value used to determine if a candidate replaces a memory cell

where, the *affinity threshold* is the system variable prepared during the initialisation process, and the *affinity threshold scalar* is a user define parameter.

3.1.5 Classification

When the training process is completed, the pool of memory recognition cells becomes the core of the AIRS classifier. The data vectors contained within the cells can be de-normalised or left as is for the classification process. Classification occurs using a k-Nearest Neighbour approach where the *k* best matches to a data instances are located and the class is determined via majority vote.

3.2 Algorithm parameters

As mentioned, the AIRS algorithm has a number of user configurable parameters for fine-tuning the training schedule to specific problem domains. It was shown by Watkins [2] that the technique remains reasonably stable (in terms of classification accuracy) over a range of parameters on a number of standard machine learning datasets. This section provides a brief overview of the nature and specific effect each of the algorithms parameters.

Affinity Threshold Scalar (ATS) – To discuss the affinity threshold scalar, it is important to understand the affinity threshold. As discussed previously, the affinity threshold is the mean affinity between a set of antigens from the training dataset. The figure simply represents the average distance known vectors are from each other in the problem space. If the training dataset was a uniform representation of the problem space, then a vector that had an affinity (distance) less than the mean could be considered already represented, and thus replaceable. This is specifically the use of the affinity threshold.

The affinity threshold scalar provides a means of adjusting the automatic threshold by making it softer (less than the mean) or harder (more than the mean). The effect of softening the threshold causes less replacement of best matching memory cells by candidate memory cells, and the reverse is true when the threshold is hardened. Common values for this user parameter are in the range [0.1, 0.3], which is a significant softening of the mean. The effect of having the scalar too close to the mean is that too many replacements occur, thus squashing the threshold down by to a factor of 10% or 20% of the mean causes less replacements, and thus a larger, more effective classification memory pool.

Clonal Rate – The clonal rate is used in three places in the algorithm as has been mentioned. Firstly, it is used in conjunction with the hypermutation rate to determine the number of clones that a best matching memory cell can create to population the ARB pool. Secondly, it is used to determine the number of clones an each ARB can create during the ARB refinement stage. Finally, it is multiplied by an ARBs normalised stimulation to determine its resource allocation. This means that the number of ARB clones created will be in the range of [0,clonalRate]. It also means that allocated resources for an ARB will also be in this range, which has an impact on the total number of resources that should be allocated. Typical values for the clonal rate are ~10.

Hypermutation Rate – Used with the clonal rate and cell stimulation, the hypermutation rate determines the number of mutated clones a best matching memory cell can create. As mentioned, the number of clones will be in the range of [0,clonalRate] which is then increased by a factor of the hypermutation rate. It is common for the hypermutation rate to be two, causing best matching memory cells to produce double the number of mutated clones as an ARB. The intent of cloning the best matching memory cell is to seed the ARB pool to begin the ARB refinement and competition stage of the algorithm.

Total Resources – The total resources places a direct limit on the number of ARBs that can coexist in the ARB pool. Given that the amount of resources allocated to each ARB is in the range of [0,clonalRate], the total resources is expected to be somewhere around: (clonal rate · the number of desired ARBs in the pool). Common values for this parameter are 150-300.

Stimulation Threshold – As mentioned, the stopping criterion to the ARB refinement process is when the mean normalised stimulation value is above the stimulation threshold. This parameter controls the amount of refinement performed on ARBs for an antigen, and thus how closely the ARBs will be to the antigen in question. Stimulation values are commonly high, around 0.9. This means that the mean stimulation value must be quite high, that is the vast majority of the ARBs in the pool must be similar to the antigen. The range for the stimulation threshold must obviously be in the range of [0,1], given the mean also will have the same range.

Number of Initialisation Instances – The number of initialisation instance is the number of randomly selected training instances used to seed the memory pool. This parameter can be in the range [0,total training instances], and is commonly set to low values such as zero or one. The flexibility of this parameter, specifically when set to one or zero, allows the algorithm to automatically determine the number and nature of memory cell elements that makeup the classifier produced.

k-Nearest Neighbours – The kNN parameter is only used during the read-only classification stage of the algorithm. As has been mentioned, it determines the number of best match memory cells used to vote by majority on the classification of unseen antigens (data vectors). When a tie occurs in the majority vote, the class index with the lowest number is always selected, making classification deterministic (as opposed to probabilistic or stochastic tie breaking strategies). Common values for the kNN are in the range of [1,7].

4. Evolution of the AIRS Algorithm

The AIRS algorithm did not spring into existence, in fact, it is a collection of elements and processes developed for other supervised and unsupervised AIS algorithms. This section provides an overview of some of the precursor algorithms to AIRS, specifically the elements and processes borrowed from said algorithms. Also provided is an overview of some of the augmentations and research performed into the AIRS algorithm over the past few years of its history.

4.1 Precursors to AIRS

This section describes a number of the AIS algorithms that pre-date AIRS. Specifically, algorithms that contain elements and processes borrowed by AIRS or similar to those of the AIRS algorithm are discussed. This should not be considered and is not intended to be a complete review of those AIS algorithms that are discussed.

4.1.1 Immunos-81

The first approach to using immune system metaphor for a supervised learning and classification system was called Immunos-81 by Carter [9]. The system was designed with the intent of taking advantages of the features of the immune system without adhering too closely to the biological aspects and equations. The exceedingly complex system used T-cell and B-cell equivalents as well as a library of elements currently in the system. It was shown to provide good results in terms of classification accuracy on the Cleveland heart disease datasets (known data sets from the field of machine learning). An interesting feature of the technique is that it was capable of learning in real time (online), meaning it is capable of continuos learning without rebuilding the system, a feature not currently implemented or investigated for the AIRS algorithm.

4.1.2 Immune Network Theory Inspired AIS

Immune Network Theory (INT) proposes that the immune system maintains a network of cells that learn and maintain memory using feedback mechanisms. A feature of this theory is that once information is learned by the network, it is then capable of being forgotten unless the information is reinforced. Work in [12] proposes an AIS based on concepts form INT. The system maintains a population of recognition cells that respond with a stimulus when presented with an antigen, and are connected with links that represent the similarity between the cells. Here stimulation is borrowed by AIRS, where affinity is normalised in the range of [0,1] and stimulation is simply the inverted affinity. The difference here is that a stimulus is also received from a recognition cells neighbours.

The system performs clonal expansion and random mutations using a mutation rate, as was borrowed for use in the AIRS1 algorithm. Also used in the system was a network affinity threshold (NAT) that also borrowed by AIRS (affinity threshold), though here it was used to control links between recognition cells. The threshold was calculated as the mean affinity associated of all links in the network. Further, the threshold was also scaled using an *A* term (affinity threshold scalar in AIRS). The technique produced some interesting results, and was designed for unsupervised clustering and visualisation of data.

The primary problem with the INT based AIS was that it suffered from a population explosion. In [13] and [14], the system was extended to support population control mechanisms, and was called a Resource Limited Artificial Immune System (RLAIS). Also introduced with the RLAIS was the concept of artificial recognition balls (ARBs) which represents a number of identical B-cell (recognition cells) in the network. The system is configured to allow a certain number of individual B-cells, which are divided by and competed for by the ARBs, where the higher the stimulation of the ARB, the more B-cell resource it is allocated. When an ARB losses all of its resources, it is considered not a useful representation of the training data, and is removed from the system. Finally, the last change made was that the NAT was calculated once at the start of the run and kept constant, rather than recalculated each algorithm iteration.

4.1.3 Clonal Selection Inspired AIS

Clonal selection theory is the idea that those cells that are effective at recognising pathogenic material are selected (in a Darwinian sense) to survive and propagate. Work

in [15] devised a technique called the clonal selection algorithm (CSA), which was revisited in [16] and renamed CLONALG which was based on the clonal selection theory. The technique uses elements of the affinity maturation process of the immune response to maintain a population of cells capable of learning a desired problem. The CLONALG algorithm was shown to be similar to that of evolutionary strategies (ES) from the field of evolutionary computation, though CLONALG searched by blind cumulative variation and selection by cloning, mutating and advancing best match recognition cells.

The process of cell stimulation, clonal expansion, stimulation proportionate somatic hypermutation, and recognition cell selection, and replacement are all elements used in the functioning of the AIRS algorithm. The CLONALG algorithm was shown to be a useful AIS algorithm, successfully applied to a number of machine learning and artificial intelligence problems such as binary character recognition, multimodal function optimisation and the travelling salesperson problem (TSP).

4.2 AIRS Research

The focus of this review of the AIRS algorithm has focused on version 2 (AIRS2). This version of the algorithm has been shown [7] to be a simpler, computationally more efficient version of the algorithm and is thus the standard AIRS implementation. The first version of AIRS (referred to as AIRS1) was very similar, though contained a few minor but important differences from the AIRS2 algorithm discussed.

4.2.1 AIRS1 and AIRS2

The AIRS algorithm as described in [1,11], and investigated and tested in detail in [2] treated the ARB pool as a persistent resource throughout the training scheme, rather than a temporary resource for each antigen as in AIRS2. This means that ARBs left over from previous ARB refinement passes (for past antigens) are maintained and participate in the competition for limited resources. The effect as mentioned in [10] is that the algorithm spends more time rewarding and refining ARBs that belong to the same class as the antigen in question. To accommodate a similar amount of training focus, the *stimulation value* user parameter is raised in AIRS2 from that used in the AIRS1 algorithm.

AIRS1 also permitted the class of generated clones to be mutated. Given that only clones of the same class as the antigen are considered in the ARB pool, the need for class mutations was also removed in the AIRS2 algorithm. The ARB pool was transformed into a temporary resource, only required to be population when working with an antigen, then cleared once training on the antigen is complete making it ready for further training.

Another important difference between AIRS1 and AIRS2 is the manner in which clones are mutated. The original AIRS1 algorithm uses a user defined *mutate rate* parameter to determine the degree to mutate a produced clone, and simply replaced attribute values with randomly generated values within the attributes normalised range. AIRS2 introduced the concept of somatic hypermutation where the amount of mutation a clone receives is proportional to its affinity to the antigen in question. This mutation scheme allows tight

search when affinity is great and a wider search area when clones are a distance from the antigen.

Results between the two algorithms were shown [10] to be quite similar in terms of classification accuracy. Besides the benefits of simplification and improved computational performance, the AIRS2 algorithm was also shown to provide better generalisation capability in terms of improved data reduction of the training dataset.

4.2.2 Extensions to AIRS

AIRS had had been shown to be a successful classifier on a broad set of well known classification problems with small numbers of classes. A study was undertaken [6] to investigate the performance of AIRS on problems with many multiple classes and an increased number of features (attributes), compared to the Learning Vector Quantisation (LVQ) algorithm. Artificial problem domains with three, five, eight and 12 classes were evaluated, as were six common problems from the field of machine learning. In just about all cases for the common real-world problems, AIRS out-performed LVQ configured with a similar number of elements as AIRS, as well as an optimised version of the LVQ algorithm. In the case of one of the datasets (credit card classification problem), AIRS achieved results better than the best classification results known at the time for the problem (to the authors knowledge).

Work in [17] investigated the effect on AIRS of introducing additional irrelevant features to classification datasets. It was speculated that the performance of the technique would decrease given its reliance on the Euclidean distance measure. Results were unexpected and indicated a small drop classification accuracy on the tested problem domains. A comparison of results with LVQ showed that LVQ was capable of out-performing the AIRS technique when initialised with the same number of codebook vectors discovered by AIRS, though was not capable of doing so when configured independently.

A number of augmentations to the AIRS algorithm have been suggested and tested in previous research. Some issues addressed in [18] include the handling of ties during classification, the restructuring of the ARB pool, and resource allocation. While focused on augmentations of version one of the AIRS algorithm, the results are still interesting. Four additional alternative tie handling methods for classification were proposed and tested in addition to the standard AIRS tie handling technique:

First labelled first served (traditional AIRS technique) – In the case of a tie, select the class with the lowest assigned identification number (first encountered)

Sum of affinities – Select the class with the highest sum affinity

Selection based on class proportions – A probabilistic approach that would use the number of instances of each class to determine probabilities

Include more memory cells – Continue to add memory cells until the tie was resolved (uses a new parameter k-additional which defines the maximum number of additional cells to include)

First come first served – Select the class of the highest stimulated cell

Interestingly the probabilistic approach "Selection based on proportions" performed the best on the noisy yeast machine learning dataset, and the "sum affinities" approach which logically appear the most sensible performed relatively poorly. The alternative ARB pool structures were centred around the assumption that the fact that ARBs of different class as the antigen in the pool had little impact on training and the resulting classifier. One alternative resource allocation scheme was also tried where resources were allocated based on the class proportions in the ARB pool (again for AIRS1). Results were inconclusive on the E. coli and yeast machine learning dataset.

The issue of using different distance measures for calculating affinity was addressed in [19]. Euclidean distance is traditionally the distance measure used in AIRS algorithm for calculating affinity and stimulation scorings. This requires that all attributes be converted to numeric (real) values, which has no meaning for nominal and potentially less meaning for discrete attributes. Twelve alternative distance measures were proposed for use in AIRS and evaluated, each capable of producing values in the require range of [0,1]. The measures used specifically catered to the nature of attributes, and were tested on common machine learning datasets with a range of numeric, nominal and discrete attribute types. Results indicated that by using more natural and arguably useful measure of comparison, AIRS can achieve better classification accuracy.

4.2.3 The Power of AIRS

At least two studies have been performed into why exactly the AIRS algorithm is so successful. In [20], it is hypothesized that the power of AIRS comes from the manner in which the algorithm derives candidate memory cells for the pool that eventually make up the resulting classifier. The random process for generating ARBs was replaced with a standard elliptical probability distribution function (PDF) and analysed. A Histogram was then prepared describing the distances of mutated clones from their parents. These histograms were prepared and evaluated for a number of common machine learning datasets.

To test the proposed hypothesis, the process of randomly generating ARBs was replaced with one of three different distribution functions based on the histograms observed. The results showed on the most part that the difference between using the different functions was not statistically significant. Further, the results lead to the speculation that the power of AIRS does in fact not come from the manner in which samples are generated, but rather from the manner in which the system selects, replaces and maintains members of the memory cell pool.

This second hypothesis regarding the memory cell selection decision making process within AIRS was investigated empirically in [21]. A number of alternative decision

making elements were proposed and tested along with the standard technique (previously described), these were as follows:

1. **Mod0** – The original AIRS1/AIRS2 scheme
2. **Mod1** – Same as Mod0, only memory cells are never delete, only added
3. **Mod2** – Same as Mod0, only when a memory cell is admitted, it always replaces the previous best match
4. **Mod3** – The candidate memory cell is always admitted, though never replaces the previous best match

The results showed that Mod1 ended up with a single memory cell for each class, whereas Mod1 and Mod3 ended up having a memory cell for each antigen it was exposed to. In terms of classification accuracy, Mod1 and Mod3 produced results similar to that of Mod0, though given the data reduction capability of the original system; it was clearly the better approach.

4.2.4 Parallel AIRS

The final interesting area of research for the AIRS algorithm is work into exploiting the parallelism inherent in the techniques base metaphor. Few AIS algorithms exploit the distributed nature and parallel processing attributes exhibited in the mammalian immune system. Work in [22] proposed and tested a version of the AIRS algorithm designed for distribution across a variable number of processes. The approach to parallelising AIRS was simple, involving the following steps in addition to the standard training scheme:

1. Divide the training data set into *np* number of partitions, where *np* is the number of desired processes running AIRS
2. Allocate a training partitions to processes and prepare memory pools
3. Gather the *np* number of memory pools
4. Use a merging scheme for creating a master memory pool for classification

Results showed that as long as the dataset was not partitioned too extensively, then a speedup could be observed by running AIRS in parallel, whist achieving similar levels of classification accuracy. A simple concatenation scheme was used which caused some loss in the data reduction benefits of AIRS, so alternative merging strategies were proposed and tested. Merging approaches using affinity between memory cells provided results that were inconclusive, and was marked as work for future research. It should be noted that the parallel version of the technique can be used on multi-processor machines via threads (see the WEKA implementation overview in section 9.).

5. Analysis and Extensions

The AIRS algorithm is a young (approximately just over three years old), and reasonably complex algorithm. Given the excellent results reported for a wide range of classification problems, the algorithm both has room for improvement and disserves investigation. This section analyses the algorithm from a design goal perspective to reiterate some of the strengths and identify some of the limitations of the algorithm. A number of AIRS specific statistics are proposed and discussed for use analysing the algorithm. Finally, a number of extensions and simplifications to the algorithm based on the identified limitations are discussed.

5.1 Expectations and Limitations of AIRS

For a list of design goals or expectations of an AIRS or AIRS like immune-based supervised learning / classification system, a good place to start is [9] that lists design goals used for the development of the Immunos-81 system. They were as follows:

1. Easily understood internal representation
2. Ability to generalise from input data
3. Predictable training times
4. Online learning
5. Potential to act as an associative memory
6. Acceptance of continuous and nominal variables
7. Capacity to learn and recall large numbers of patterns
8. Experience-based learning
9. Supervised learning

5.1.1 AIRS and Design Goals

The mentioned set of design goal seem reasonable for an AIRS-like supervised learning system, therefore it is beneficial to evaluate AIRS against each goal to aid in identifying potential limitations.

5.1.1.1 Easily understood internal representation

The benefit of having an easily understood representation is that it allows a user to look at the classification decisions made by the system and understand directly why a particular action was taken. An opaque system creates a situation where the user has little or no understanding of how the system arrived at a decision. Some artificial neural network architectures suffer from this problem. AIRS uses natural or real vectors to represent data in the same manner as data is represented in the domain. AIRS clearly meets this design goal, though it is common during the initialisation process in AIRS to normalise the data vectors, though (as it will be shown in section 5.3) this may not necessary.

5.1.1.2 Ability to generalise from input data

To generalise is to draw a specific case from a more general case. In terms of classification, AIRS uses exemplars to represent the general case and determines the specific case (classification) based on a best match or majority vote from the top k best

matches. Moreover, the exemplars are representative of one or more training instances, meaning that the number of exemplars is less than that of training instances. This data reduction feature has been observed to be up to approximately 50% or more for a range for standard machine learning datasets. The minimum requirement for an AIRS like system is to have as many or less exemplars as there are training instances.

5.1.1.3 Predictable training times

This design goal is the first design goal for which AIRS may have a problem. Although the computational complexity for AIRS has not been described, it can be estimated to be of reasonably high complexity given the repeated number of similarity (affinity and stimulation) elements involved for each antigen presented during training. Further, the algorithm consists of loops within loops within loops (resource allocation and pruning → ARB refinement → antigen presentation), which are dependent on the antigen in relation to the memory pool at the time in the training schedule. Even though computational improvements have been made on the original AIRS algorithm (AIRS2), this remains an area that needs both further investigation and improvement.

5.1.1.4 Online learning

Online learning or continuous learning refers to the algorithms ability to improve or perform further training after delivery of the classifier. LVQ and many artificial neural network algorithms are examples of algorithms that exhibit this feature. This feature has not been explored in the AIRS algorithm to date (to the author's knowledge). Some potential concerns with this goal are addressed in section 5.1.2.1.

5.1.1.5 Potential to act as an associative memory

A classification system could be viewed as a specific type of associative memory called hetro-associative (key → content). Here the key represents the data vector provided to the system, which is then matched onto an exemplar. The content returned is the classification value associated with the exemplar. AIRS clearly exhibits this feature. It is arguable whether the system can be considered auto-associative (key → key), because not only can more than one exemplar be returned, but the exemplar is rarely if ever the same as either the training data vectors or unseen test data vectors.

5.1.1.6 Acceptance of continuous and nominal variables

According to [19], the AIRS algorithm requires that all variables to be converted to numeric before being used. Extensions in [19] provide a means for handling any of numeric, discrete and nominal attributes. The problem of attribute handling in similarity or distance measures is an old problem which has been addressed and solved many times for instance based algorithm such as k-Nearest Neighbour and LVQ. This design goal should be extended to the following: "Acceptance of any attributes for which a distance useful can be devised". Given this refinement, AIRS can clearly meet this goal.

5.1.1.7 Capacity to learn and recall large numbers of patterns

This is an interesting design goal as the term large is relative and undefined. The simple answer to whether AIRS can meet this goal is yes, though in practicality it appears

problem dependent. It is fair to say that AIRS will find some arrangement of exemplars to satisfy the training data it is exposed to. A problem of time complexity may arise when the number of exemplars increases beyond a reasonable level. This is addressed in section 5.3.5.

5.1.1.8 Experience-based learning and Supervised Learning

The last two design goals are quite vague and similar, in fact you can't have one without the other. The first: experience based learning, indicates that the algorithm learn from its experience with the problem domain (training data), and the second: supervised learning, indicates that the learning it performs is supervised, meaning both input and desired output are known at training time. AIRS is a supervised learning algorithm that learns from experience.

5.1.2 Additional Design Goals

The design goals discussed are reasonably general, and provide a good starting point for an AIS based AIRS-like classification system. This section proposes a number of additional design goals and attempts to answer the degree to which AIRS addresses them.

5.1.2.1 Prior Knowledge

The question of what prior knowledge is required for AIRS is an interesting question. On the surface, it may appear that little or no prior knowledge is required to run the AIRS algorithm. This assumption comes from the results observed on a broad range of standard classification problems with factory or close to factory algorithm parameters.

Firstly, the algorithm indicates that all training data vectors are normalised during initialisation. Normalisation requires the bounds of each variable to be known prior to training. This is fine, because the training set is finite and known at training time. If the resulting memory cells produced by the algorithm in the classifier are kept normalised then the bounds of all attributes seen during classification are required prior to training. This is commonly not a concern for most if not all toy machine learning classification problems where the entire data set is small and known. It becomes a concern when working with large real-world dataset and data streams where continuous learning is applied.

In the highlighted cases, the bounds of the attributes can be speculated at, but may not be known. As will be shown in section 5.3, AIRS can be re-deigned to get away without requiring normalisation of training vectors. A problem still arises during ARB generation. Here, the degree of mutation a cloned ARB receives is proportional to its stimulation. The mutation amount becomes the inverted normalised stimulation proportion of each attributes range.

The final place where the bounds of the input space are required is in calculating the maximum distance between any two data vectors, which is in turn used to normalise all distance values (used for affinity, and stimulation throughput the algorithm). These issues are somewhat addressed in section 5.3.

This design goal indicates that the domain specific knowledge should be minimal. The question as to whether knowing or intelligently speculating at attribute bounds *a priori* is too much prior knowledge can be considered problem and application dependent, though it is desirable to have a version of AIRS where this is less or not even a concern.

5.1.2.2 Classification Accuracy

A critical design goal of an AIRS-like classification system is its classification accuracy. This accuracy should be measured using a tested testing procedure such as cross-validation and results need to be averaged over tens, hundreds or even thousands or run executions to give somewhat statistically safe results. (It should be mentioned that the majority of reported results are mean accuracy values over 3-10 tests, typically in an attempt to reproduce previous experiments for comparison) Further, a mean and standard deviation of overall classification accuracy is less than meaningful for many machine learning datasets given disproportional class distributions. At a minimum a confusion matrix should be provided and used to give a more accurate indication of classifier usefulness, other than accuracy.

In classification mode, AIRS is used in a similar manner to the k-Nearest Neighbours (kNN) algorithm. A suitable design goal is to require that an AIRS-like supervised learning algorithm provide equal or better classification performance than kNN. Results should also be competitive with those of LVQ, a similar algorithm in classification function, which is known to both produce good classification results and is computationally efficient.

5.1.2.3 User Parameters

Most supervised classification algorithms require one or more user parameters, and AIRS2 is no exception with nine parameters (including random number generator seed). The AIRS1 algorithm had an additional two parameters, taking the total to 11. User parameters are a beneficial feature in that they allow the user to tune the algorithm for a specific problem domain. The primary problem is that changes to parameters need to be predictable. This problem becomes more of a concern as the number of parameters increases.

A consolation is that many of the results reported of the AIRS algorithm have been found using AIRS with factory configuration, and the fact that good results have been achieved with these settings across a broad range of problems raises an interesting question. How many of the parameters are actually critical to the functioning of the AIRS algorithm? This problem will be somewhat addressed in section 5.3. It is clear that a suitable design goal for an AIRS-like algorithm is to minimise the number of user configurable variables, and have the effects of those variables that are left simple and predictable.

5.2 AIRS Specific Statistics

From a review of the AIRS literature, one area that has seen little or no attention is the topic of user feedback regarding the training process and state of the classifier produced from the training process. This section proposes a number of AIRS specific statistics that may be useful for both tunning the algorithm for a specific problem domain, as well as

providing some insight into classifiers produced by the algorithm. It is important to note that all statistics mentioned in this section have been implemented in the WEKA implementations of the AIRS algorithms (See section 9.).

5.2.1 Training statistics

Statistics gathered during the algorithm training process can be used to gain insight into the functioning of the system and be used to refine algorithm performance either in terms of classifier quality or algorithm runtime. Of specific interest during the classification preparation process are statistics that provide an indication the effects of user-defined parameters.

Number of Clones – The mean number of clones produced by memory cells and ARBs during training provides a useful indication of how much local searching is occurring. Mean scores closer to the maximum (*clonal rate* for ARBs and (*clonal rate* · *hypermutation rate*) for memory cells) indicate that stimulation scores are higher, perhaps indicating that the system may be over-learning antigens requiring the *total resources* to be reduced or the *stimulation threshold* decreased. The reverse could be the case if the mean is closer to zero.

Allocated Resources – The mean number of allocated resources provides an indication of the amount of ARB pruning that may be occurring during training. A mean closer to the *total resources* may indicate that the total resources may not be high enough, where as a mean a distance from the total may indicate more pressure is required – that is a lower *total resource* value.

ARB Pool Size – The mean number of ARBs in the pool provides further insight into the resource allocation and competition for limited resources. Smaller pool sizes indicate perhaps not enough cloning or too much competition for resources; whereas large pools cause the training scheme to run slower and perhaps require a lowering of the clonal rate or an increase in the competition for resources.

Refinement Iterations – The mean number of ARB refinement iterations provides an indication of the amount of local search performed for a given antigen. A large mean indicate the step may be too computationally expensive and may result in the system over-learning the training set resulting in lower classification accuracy.

5.2.2 Classifier Statistics

The classifier is simply a pool of data vectors (memory cells) that represent a generalisation of the training dataset. Of interest of this pool of data vectors are measures of class breakdowns and vector usefulness.

Class Breakdown – A telly of the number of memory cells that belong to each of the known classes. Provides an overview of the coverage of the problem space, and could be compared to a similar tally of the training data to gain an indication of usefulness or skew in the generalisation that has occurred.

Percentage of Data Reduction – The amount of data reduction in the resulting classifier could be used in conjunction with classification accuracy as a measure of algorithm usefulness. This measure effectively conveys the amount of generalisation that has occurred, and when paired with classification accuracy on the training data provides an indication of quality of the generalisation. The data reduction could be a percentage calculated as:

$$data\,\mathrm{Re}\,duction = \left(1 - \frac{numMemoryCells}{numTrainingData}\right) \cdot 100$$

Equation 10 - Calculation for data reduction measure of resulting classifier

It should be noted that this data reduction measure is not original. It was used in [7,10] as a means of comparing AIRS1 and AIRS2. What is suggested here is that the measure becomes a standard when reporting results of AIRS.

Memory Cell Usefulness – Calculating a histogram of hits against the training or test data set provides an indication of the usefulness of the memory cells in the classifier. The hit counts could further be broken down into correct and incorrect classifications and perhaps used to allow memory cells to "change sides" or change their classification outputs dynamically at runtime / classification-time. Further, a histogram against the training data could be taken and all memory cells that have zero hits could be removed from the classifier – providing a refinement to the data reduction process.

5.3 Restructured AIRS

From reviewing the literature regarding the AIRS algorithm, as well as analysing AIRS from the perspective of reasonable design goals, a number of extensions, simplifications and restructurings to the AIRS algorithm can be proposed. The propositions vary in detail from major changes to the algorithm itself, to simple reinterpretations and implementation specific details. The intent of this section is to address a number of raised issues regarding the AIRS algorithm in an attempt to better match some or all of the proposed design goals to both make the algorithm simpler from a user perspective whilst achieving results similar, equal or better than those of the base AIRS (AIRS2) algorithm implementation.

To recapitulate the main elements of AIRS that are immune-inspired from the field of AIS, the following list is provided:

1. Affinity Maturation – The process used to adapt the system to a presented antigen
    a. Clonal Expansion – The affinity proportionate means of determining the amount of local exploration (number of samples)
    b. Somatic Hypermutation – The stimulation proportionate means of determining the amount of local search (variation from parental unit)
    c. Clonal Selection – The affinity determined means of selecting a candidate recognition cell and integrating it into the pool (selection and potential replacement)

2. Long lived Recognition units – The ARB representation and maintenance of the memory pool itself that ultimately becomes the resulting classifier

At its simplest the algorithms design is to prepare a set of exemplars by generating variations of existing best match exemplars and using a selection and replacement policy to integrate newly generated exemplars into the managed pool. Obviously, there is more to the sample generation process such as refinement and resource allocation for pruning, but these elements detract from the algorithms core function. Interestingly, as was previously shown, the means of sample generation have little effect on the resulting classifier. Replacing the entire sample generation process with a simple function for generating exemplars within a constrained radius is attractive for a number of reasons. This will be revisited shortly, first it is important to reiterate and understand some implementation details of AIRS.

5.3.1 Normalisation

As mentioned, the training dataset must be normalised during initialisation. Why must the dataset be normalised? From a detailed review of the algorithms pseudo code, it appears that the primary reason for this requirement is the mutation process. Data normalisation is used in the algorithms design to enforce distances measures to be in the range of [0,1], though as was shown can be avoided here by simply dividing the distance by the maximum possible distance.

The mutation process is where the inverse stimulation value is used as a proportion of mutation for each attribute in a clone, with the clones current attribute value as the centre, and the inverse stimulation value as the range for the new mutated attribute value. It could be argued that normalising for this reason alone is insufficient. The same mutation effect can be achieved by knowing the attribute range in the normalisation set and setting the mutation range as (attribute range · inverse stimulation value). This removes one of the elements of prior knowledge identified in AIRS.

5.3.2 Normalised Distance

All distance measures calculated during the training scheme (not classification) are normalised, and are then used as affinity or inverted for stimulation. Why is it necessary to normalise distance measures? The following lists all the elements of the algorithm dependent on normalised distance measures:

1. Affinity Threshold calculation – used with the user defined affinity threshold scalar to determine when a candidate memory cell can replace the previous best match memory cell
2. Affinity for memory cell selection replacement – required because of affinity threshold (circular argument)
3. Stimulation – for memory cell generation (number of clones) and mutation (mutation range) after best match memory cell is selected

Interestingly all resource allocation, stopping criterion and ARB generation and mutation during the ARB refinement uses stimulation values that are normalised against all other ARB stimulation values. Although the pre-normalised stimulation values are calculated using normalised distance values, distance normalisation at this point is clearly not required.

The clonal selection element of the algorithm has a circular argument requiring normalised distance values. By calculating the affinity threshold using un-normalised distances measures and calculating affinity in this process in the same manner, normalised distance measures are no longer required at this point. Regarding the affinity threshold scalar parameter, as mentioned it is simply a squashing term and normalised distance scorings have no bearing on the resulting memory cell replacement decision-making process.

This leaves the seed ARBs generated from the best matching memory cell. In the AIRS algorithm, the number of clones generated at this point is calculated using Equation 6, which requires two user parameters. An easy solution is to simply require the user to indicate the number of seed ARB clones for the ARB pool each iteration which replaces two scale parameters with one simple numeric parameter (easier to understand). Regarding mutation, the degree of variation could be determined by the number of seed ARBs specified. More ARBs naturally means more room for exploration, and thus mutation range could take advantage of this.

This is an arbitrary solution, and it is clear more thought is required in a more effective mutation scheme. What is clear is that it is possible to further remove distance normalisation from the algorithm, another element of complexity that requires prior knowledge of the problem domain. This change becomes more attractive when the mutation scheme is replaced.

5.3.3 Removing Unnecessary Parameters

One of the strengths of AIRS is its ability to discover or determine the architecture (number of exemplars) automatically. A review of the literature revealed that the number of seed memory cells is commonly one or zero. Give these two points, it seems logical to remove the parameter from the algorithm, or at the very least leave it an optional, but non-required parameter.

The resulting effect of this change is that the first antigen encounter from each class will automatically become a member of the memory pool, at least until it is potentially replaced. A minor flaw in the original pseudo-code for the algorithm is that even when a best match memory cell is identical to the antigen (as in this case), the ARB refinement process continues. The effect is that the affinity is zero and the stimulation is one. This means that one clone is produced (the original best match becomes an ARB), the clone is not mutated, and the stopping condition for ARB refinement is met the first time it is checked. Finally, the single ARB clone is selected and ends up replacing the original antigen cloned memory cell. A small change with an impact on computational complexity

is to simply not run the ARB refinement loop when an antigen becomes a memory cell, or when affinity is zero between a memory cell and an antigen.

Another small change along the same lines is to change the algorithms response when the best matching memory cell belongs to another class. In this case it means that the system is miss-classifying the instance to begin with. To then generate, mutate and refine clones of the misclassification makes little sense. In this case the antigen could be simply directly added to the memory pool, or the seed used to start the ARB refinement process could be a version of the best matching memory cell with its class adjusted to that of the antigen in question.

A second parameter that can be removed with little consequence is the number of antigens to use when calculating the affinity threshold. For small problems, it is reasonable to simply calculate the mean distance between all instances in the training dataset. In larger problems or in online learning where this is not feasible, the affinity threshold can be calculated as a running mean, potentially initialised from a reasonably sized sample from the domain.

The affinity threshold is, as has been mentioned, the control over when candidate memory cells replace best match memory cells. Little research has been performed into this simple yet critical element of the AIRS algorithm. It would be useful to investigate the effect of using dynamic (running mean), and class-based affinity threshold measures in the algorithm. Further, would also be useful to evaluate alternative replacement decision making strategies and their impact both on data reduction and classification accuracy.

5.3.4 Sample Generation

As has been shown in previous work, the source of power of AIRS appears to be the memory cell selection and management process, rather than the cell (sample) generation process. This is great news, given that the majority of the computational complexity of the AIRS algorithm pertains to sample generation. The first simple step is to replace sample generation with a random sample generation function constrained both in the number of samples and distance of samples from parent to child. This constraint could be (not does not have to be) imposed in some way by the affinity or stimulation between the ARB and antigen.

Regardless of the ARB generation process, it is logical to remain consistent with the basic function of AIRS. That is, to first generate a seed batch from the original best match memory cell, and then to refine the ARBs to some degree. A basic implementation is to simply run one ARB refinement iteration, ensuring that the number of clones for each ARB is approximately half that of the number of clones generated from the best match memory cell to seed the ARB pool (standard AIRS parameters). It is speculated that this simple approach would be enough to provide sufficient variation of the original best match memory cell to allow AIRS to achieve similar levels of classification accuracy and data reduction. This conjecture is based on the results from [20], and the fact that this version of the process provides a more AIRS-like sample generation scheme.

5.3.5 Classifier Structure

In the standard AIRS implementation, the computational complexity for classification (assuming a k of one) is O(n), where n is the number of exemplars. In the case of both a very large classifier and continuos learning system, this is less than desirable, especially if the exemplars are managed externally from the system (a database for example). It is possible to restructure both the pools used during training (ARB pool and memory cell pool), as well as the final classifier to use a more attractive data structure, such as a tree of some description. Again, as with vector distance measures, this is an area of research that has seen a lot of attention both in terms of the kNN and LVQ algorithms. A simple change is to implement a known more efficient data structure to speedup all matching processes in AIRS.

6. Further Work

The AIRS technique has been shown to be a powerful and successful machine learning algorithm, though a technique with perhaps an unnecessary amount of complexity. A number of extensions, simplifications and changes to the standard AIRS (AIRS2) algorithm were proposed based on an analytical and theoretical understanding of the technique. What is clearly required is an investigation into the effects on classification accuracy and data reduction capability of the technique with some or all of the proposed changes implemented. It should be noted, that work on this investigation has already begin, using the WEKA implementation (discussed in section 9.) as the foundation.

Analysing and researching the history of a technique is a beneficial exercise for gaining a strong grounding in the technique being investigated as has been demonstrated by this work. To this end, it would be a useful exercise to perform a similar investigation and analysis on other successful AIS based algorithms, specifically those techniques which have already been or can easily be converted to the primary interest of the author, that is supervised learning for classification. Some suggestions include Immunos-81, CLONALG, AINE and the ai-NET algorithms. Further, it may be beneficial to investigate the main themes and theories of the biological immune system for both a stronger grounding in AIS terminology, as well as inspiration for further developments in the field of artificial immune systems.

It should be noted, that at least two areas of research regarding AIRS were not investigated fully in the production of this work. The first relates to parallel AIRS, specifically work by Andrew Watkins [23] (unpublished dissertation). The second is the use of AIRS for document classification by Julie Greensmith [24,25]. The reason for these omissions was the inability to get a hold of the related works within a reasonable time period before this work on investigating the AIRS algorithm was to be completed. It is expected that a review of this mentioned work would provide further insight into the AIRS algorithm in terms of both function and application.

7. Conclusions

This work represents an entry point for the author into the field of artificial immune systems, and specifically the AIRS algorithm. The intent of this work was to demonstrate

competence with the AIRS technique both in terms of history, algorithmic function and application. Further, given some competence, the intent included proposing potential extensions to the technique. These intentions of this work were achieved successfully. A complete (as the author knows it) history of the AIRS algorithm was provided with a focus on research that was of fundamental or of interest. Not only were the technical elements of the technique discussed, but in addition, a professional-level, tested implementation of the main technique was provided as a plug-in for the widely used and known WEKA machine learning work bench application and application programming interface.

Finally, a number of potential limitations and areas for improvement regarding the AIRS algorithm were identified. These limitations were then addressed through the proposition of a number of algorithm simplifications and extensions both original and borrowed from other research. Although the effect of said extensions was speculation and conjecture, it is expected that the implementation of one or more may produce a variant of the AIRS technique or a new technique entirely that is not only more usable (in terms of application and or implementation), but is likely to provide as good or better classification accuracy.

8. Appendix – Reported Results

This section provides a summary of results reported with the various AIRS version for common machine learning datasets. The results reported here should not be considered complete, rather the intent of this section is to provide both a summary of potentially good algorithm parameters and an expectation of results for the technique that can be used as a measure of comparison.

8.1 AIRS1

The following results were reported in [2] for the AIRS1 algorithm.

Dataset	Cross-validation	Total Tests	Seeds	Res.	Stim.	Mut.	ATS	kNN	Test Accuracy
Iris Plants	5	3	1	200	0.9	0.1	0.2	7	96.67%
Ionosphere	First 200 train Last 151 test	10	1	500	0.8	0.1	0.2	3	94.92%
Pima Diabetes	10	3	1	200	0.9	0.1	0.2	9	74.09%
Sonar	13	10	1	200	0.9	0.1	0.2	1	83.99%

Table 1 - AIRS1 parameters and results for four common datasets

The following results were reported in [6] for the AIRS1 algorithm. Algorithm settings for the results were not provided, though were indicated to be "as shipped". All tests were reportedly executed using 10 fold cross-validation and were taken as the average result over three tests.

Dataset	Test Accuracy
Balance-scale	96.7%
Pima diabetes	74.1%
Wisconsin breast cancer	97.2%
Credit.crx	85.7%
Ionosphere	94.9%
Arrhythmia	59.5%

Table 2 - AIRS1 results for six common datasets

8.2 AIRS2

The following results were reported in [10] and were reportedly arrived at by using the same configuration and setup as was used in [2]. The results were provided as a comparison between AIRS1 and the newly proposed AIRS2 technique.

Dataset	Test Accuracy
Iris Plants	96.0%
Ionosphere	95.6%
Pima Diabetes	74.2%
Sonar	84.9%

Table 3 – AIRS2 results for four common datasets

9. Appendix – Overview of WEKA implementation

WEKA is a machine learning workbench [26] written in the Java programming language. It provides a large number of common classification, clustering, attribute selection algorithms as well as visualisation tools, algorithm test schemes and data filtering tools. WEKA provides a number of interfaces for making use of the tools and algorithms provided such as a command line interface, a data exploration interface, an algorithm test and comparison interface, a workflow interface, and finally a programmer level application programming interface for integrating WEKA functionalities into standalone allocations. WEKA has been made open source, allowing academics and industry to extend the platform by adding algorithm and tool plug-ins for the platform.

The AIRS algorithm has been shown to be a successful classification algorithm for a broad range of machine learning problems. As apart of this work investigating the AIRS algorithm, an implementation of AIRS was prepared and tested for WEKA. As far as the author knows, there does not exist an implementation of the AIRS algorithm for professional and academic level application that is both documented and open source.

The provided implementation of AIRS for WEKA offers the following advantages:

1. Standardised programmatic and user interface
2. Java implementation (platform independent)
3. Open-source, taking advantage of refinements and improvements from contributors
4. Allows AIRS to be made available to the wider user base of WEKA users (both academic and industrial)
5. Provides a convenient, consistent, proven and tested platform for algorithm benchmarking, testing and comparison
6. Take advantage of the wide array of classifier-specific performance measures, visualisation tools, standard datasets and data filtering and preparation tools
7. Prepared models (including classification models), as well as algorithm configurations can be saved externally, reused and distributed

The WEKA implementation provides three variants (at the time of writing) of the AIRS algorithm. These include AIRS1, AIRS2 (canonical AIRS) and parallel AIRS. All AIRS implementations are located in the package “weka.classifiers.immune” and are accessed as such in the WEKA user interfaces from the algorithm selection drop-down.

9.1 Common Features

The implementation provides statistics mentioned in 5.2, as well as some additional useful statistics. Both sets of statistics, namely the training summary statistics and the classifier statistics are provided in the WEKA interface automatically, or can be retrieved programmatically by executing the toString() method on a prepared algorithm instance. The following provides an overview of the provided statistical information:

1. Training Statistics
    a. Affinity Threshold – The system calculated affinity threshold value
    b. Total Training Instances – Total data instances used for the training process
    c. Total memory cell replacements – The total number of times a candidate memory cell replaced the previous best matching memory cell
    d. Mean ARB clones per refinement iteration – The average number of ARB clones produced each ARB refinement iteration
    e. Mean total resources per refinement iteration – The total number of allocated resources (after pruning) allocated per ARB refinement iteration. For AIRS1, this is the sum of allocated resources for all classes
    f. Mean pool size per refinement iteration – The average ARB pool size for each ARB refinement iteration
    g. Mean memory cell clones per antigen – The average number of memory cell clones made of the best match memory cell to seed the ARB pool for each training data instance
    h. Mean ARB refinement iterations per antigen – The average number of ARB refinement iterations per presented training data instance
    i. Mean ARB prunings per refinement iteration – The average number of ARBs pruned from the ARB pool per ARB refinement iteration
2. Classifier Statistics
    a. Data Reduction Percentage – The percentage reduction in the number of memory cells compared to the number of training data instances
    b. Classifier Memory Cells – A breakdown of the number of memory cells that belong to each class

All AIRS implementations are provided with the following features and augmentations

- Data instances with missing values are removed from the training data set, though are supported when performing classification with the resulting model. This is achieved by ignoring attributes with missing values when calculating the affinity measure
- Both numeric (real) and nominal (unordered discrete) data instances are treated correctly when calculating the affinity measure. In the case of nominal values, a binary matching process is used for equal (zero) and not equal (one).
- During the training scheme, all cases when either a data instance is converted to a memory cell or when the best matching memory cell is selected and has a stimulation of one, then the normal ARB refinement process is not executed. The reason for this is as described in section 5.

9.2 Test Results

For reference purposes, the implemented version of AIRS were tested on a number of standard machine learning datasets from [3]. All tested were executed with the algorithms default parameters, cross-validated with 10 folds and test results were averaged over 10 iterations. Although the results are somewhat similar to those results reported in AIRS literature, it is expected that through parameter tuning that improved results can be

achieved. The following table provides the mean classification accuracy scores and standard deviations.

Dataset	AIRS1	AIRS2	Parallel AIRS
Iris Plants	95.267% (1.281)	95.0% (1.0)	95.6% (0.533)
Pima Indians Diabetes	69.701% (1.215)	70.872% (1.348)	70.964% (0.79)
Sonar	69.808% (2.666)	66.01% (2.172)	64.712% (3.276)
Balance Scale	81.552% (1.127)	81.024% (0.877)	82.976% (0.687)
Wisconsin Breast Cancer	96.838% (0.289)	96.166% (0.372)	96.466% (0.203)
Cleveland Heart Disease	80.792% (1.01)	79.142% (1.452)	79.571% (1.119)
Ionosphere	87.037% (1.454)	85.128% (1.335)	84.444% (1.21)

Table 4 - Shows test results of implemented AIRS algorithms for the WEKA platform

9.3 AIRS1

Although AIRS version 1 is now deprecated, (superseded by AIRS version 2), it is provided for historical reasons primarily because of the successful results reported initially in [1,2,10]. The algorithm is a pure implementation as defined in [2] and allows the user to specify the full array of parameters. The following provides an example screenshot of the WEKA user interface configuration panel for AIRS1. All parameters are initialised with sensible default values from AIRS literature.

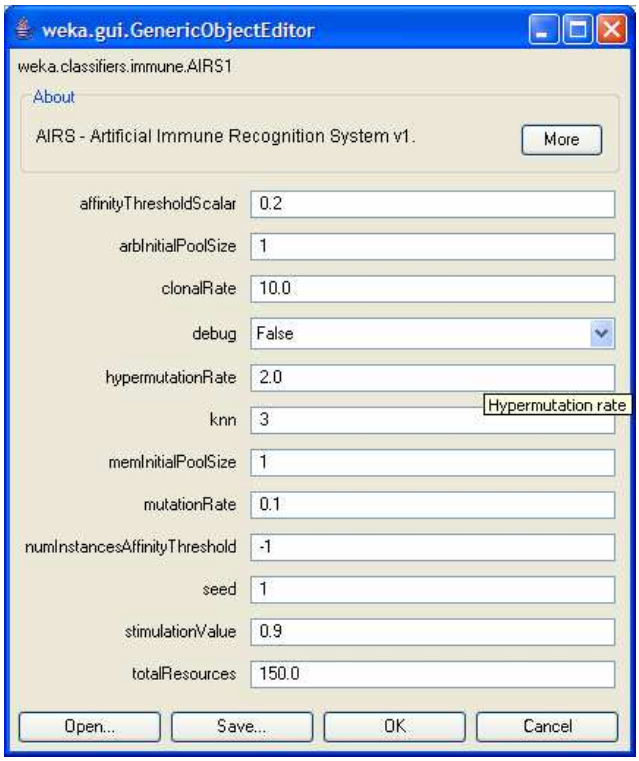

Figure 3 - Shows the AIRS1 configuration panel from the WEKA interface

9.4 AIRS2 (Canonical AIRS)

AIRS version 2 is the standard version of the algorithm, and is simply referred to as AIRS in this work. The implementation takes advantage of the reduction in computational complexity suggested in [10] and matches the description from [10] and pseudo code from [7]. The following provides an example of the configuration panel for AIRS2 in the WEKA user interface.

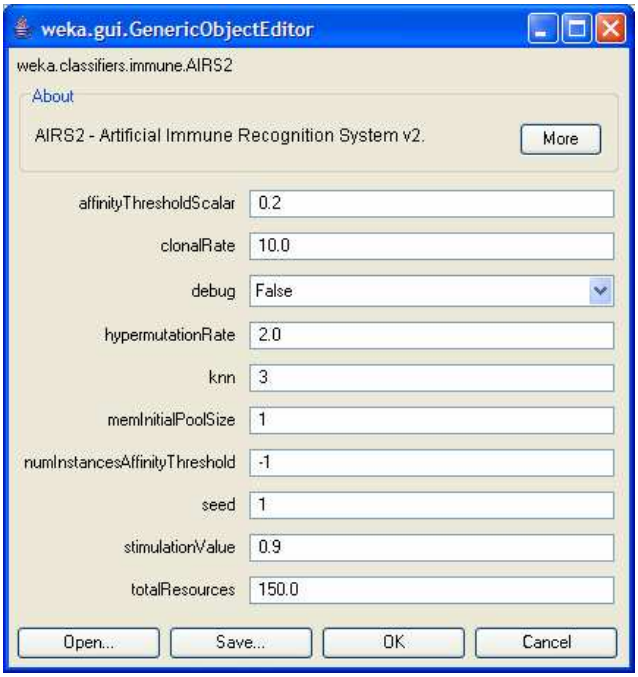

Figure 4 - Shows the AIRS2 configuration panel from the WEKA interface

9.5 Parallel AIRS

The final implementation of AIRS is the parallel version defined in [22]. Here, instead of being distributed across multiple processes, this implementation allows AIRS to be executed by multiple threads. The number of partitions can be specified, and the AIRS algorithm executed by each partition is version 2 (canonical AIRS). The parallel version of AIRS can be executed on single or multiple CPU machines; though no speed benefit is expected unless the number of specified partitions matches or is less than the number of CPUs on a multi-CPU system the system. Two means of partition merging are provided, specifically a standard concatenate, and a concatenate and prune scheme. The prune scheme first creates a master memory pool, and then applies the training data one last time. All memory cells that are not used for this final pass of best matching are removed from the pool. The following provides an example of the Parallel AIRS algorithm configuration panel from the WEKA user interface.

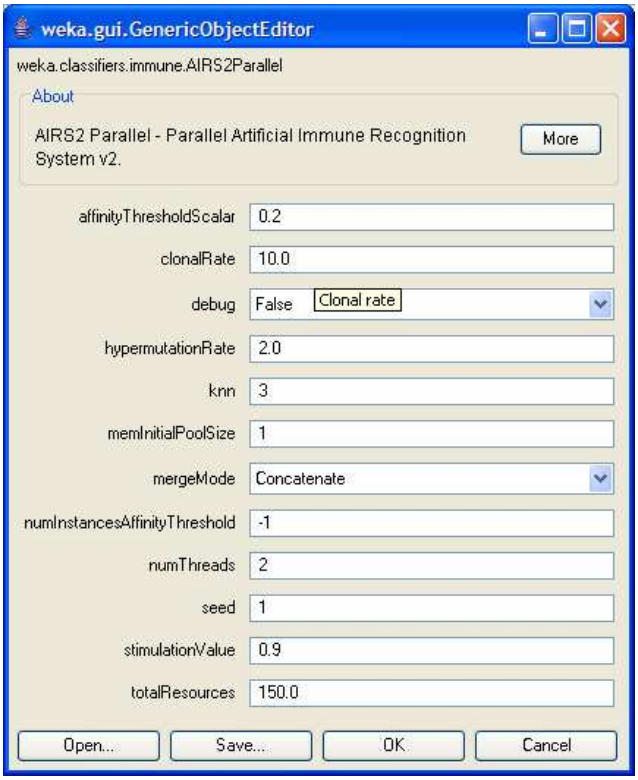

Figure 5 - Shows the AIRS2 Parallel configuration panel from the WEKA interface

9.6 Algorithm Usage

As mentioned, the AIRS implementations can be used directly from the WEKA Explorer WEKA Work Flow, and WEKA Experimenter interfaces. For the AIRS implementations to be recognised by WEKA, the AIRSWeka.jar file must be in the Java class path. The implementation was prepared with Java 5.0 (1.5), and thus the installed Java Runtime Environment (JRE) must be also be this version. Finally, the version of WEKA that the algorithm was prepared for and tested with is 3.4.3.

The following provides examples of using the three AIRS implementations from the command line on the Iris Plants dataset with 10-fold cross-validation.

```
java –cp weka.jar;AIRSWeka.jar weka.classifiers.immune.AIRS1
-S 1 -F 0.2 -C 10.0 -H 2.0 -M 0.1 -R 150.0 -V 0.9 -A -1 -B 1
-E 1 -K 3 -t data/iris.arff

java –cp weka.jar;AIRSWeka.jar weka.classifiers.immune.AIRS2
-S 1 -F 0.2 -C 10.0 -H 2.0 -R 150.0 -V 0.9 -A -1 -E 1 -K 3 -
t data/iris.arff

java –cp weka.jar;AIRSWeka.jar
weka.classifiers.immune.AIRS2Parallel -S 1 -F 0.2 -C 10.0 -H
```

```
2.0 -R 150.0 -V 0.9 -A -1 -E 1 -K 3 -N 2 -M 1 -t
data/iris.arff
```

Figure 6 - Shows examples of executing the three AIRS WEKA implementations from the command line

The following code sample provides an example application of using the AIRS2 algorithm in standalone mode. The program loads the Iris Plants dataset and performs a 10-fold cross-validation test.

```
public class SampleAIRSUsage
{

     public static void main(String[] args)
     {
          try
          {
            // prepare dataset
            Instances dataset = new Instances(
              new FileReader("data/iris.arff"));
            dataset.setClassIndex(dataset.numAttributes()-
1);
            AIRS2 algorithm = new AIRS2();
            // evaulate
            Evaluation evaluation = new Evaluation(dataset);
            evaluation.crossValidateModel(algorithm,
              dataset, 10, new Random(1));
            // print algorithm details
            System.out.println(algorithm.toString());
            // print stats
            System.out.println(
              evaluation.toSummaryString());
          }
          catch(Exception e)
          {
               e.printStackTrace();
          }
     }
}
```

Figure 7 - Shows Java code implementation of an application using AIRS2 to classify the Iris plants dataset

10. Glossary

Terms	Descriptions
Affinity	A measure between recognition cells or between a recognition cell and an antigen. Commonly implemented as a Euclidean distance measure so that the smaller the distance the greater the affinity.
Affinity Maturation	The process of a recognition cell responding to an antigen, performing clonal expansion and finally clonal selection.
Affinity Threshold	A system variable calculated during the initialisation stages of the AIRS algorithm as the mean affinity between a set of antigens from the training set.
AIN	Artificial Immune Network An AIS based on abstractions and principles from the INT of immune function. Typically AIN algorithms are unsupervised and consist of a number of recognition cells connected with links based on affinity. Examples include the AINE algorithm.
AIRS	Artificial Immune Recognition System
AIS	Artificial Immune System A class of artificial intelligence algorithm, inspired by elements of theoretical and empiric knowledge of the biological mammalian immune system. AIS algorithms are suited to difficult or intractable problem domains were more conventional techniques perform poorly.
Antibody	A feature of the immune system, produced by a recognition cell to neutralise an antigen.
Antigen	In the AIRS algorithm, represents a single training instance.
ARB	Artificial Recognition Ball In the AIRS algorithm, represents a collection of similar or identical recognition cells.
B-cell	A pathogen recognition cell of the natural immune system
Clonal Expansion	Apart of affinity maturation, where a number of clones of a recognition cell are produced to better fit an antigen.
Clonal Selection	Apart of affinity maturation, where a reduced set of mutated clones are selected for survival that have higher affinity with the antigen in question.
CLONALG	A AIS algorithm based on the clonal selection immune theory. Successfully applied to a range of problem domains such as travelling salesperson (TSP), multimodal function optimisation and character recognition.
Immunos-81	First immune system inspired supervised learning algorithm.
INT	Immune Network Theory A theory of immune function that assumes a network of connected recognition cells that learn using feedback mechanisms.

kNN	k-Nearest Neighbour. A naïve classification algorithm that uses the entire training dataset to determine the class of unseen data. AIRS uses a similar approach for classification where the best *k* matches determine the class of unseen data
LVQ	Learning Vector Quantisation A supervised learning algorithm that manipulates a fixed size pool of exemplar vectors to represent a training dataset, for classification of unseen data vectors. Devised by T. Kohonen.
NAT	Network Affinity Threshold A threshold used in AIN inspired algorithm to determine when links between recognition cells can and cannot be made. Calculated as the mean affinity between all known recognition cells. This was the basis for the affinity threshold used in AIRS.
Pathogen	A foreign l in side an organism that is potentially harmful to the organism. The goal of the immune system is to detect and neutralise pathogens.
RLAIS	Resource Limited Artificial Immune System An AIS that introduces a population control mechanism using ARBs that are allocated a B-cells based on affinity with an antigen. ARBs that are then determined to have an insufficient number of resources are removed from the system.
Somatic Hypermutation	In the AIRS algorithm, a mutation process of ARB clones where the degree of mutation is proportional to the degree of affinity between the ARB and the antigen
T-cell	A recognition cell in the natural immune system

11. Bibliography

[1] Andrew Watkins and Lois Boggess, "A New Classifier Based on Resource Limited Artificial Immune Systems," *Proceedings of Congress on Evolutionary Computation,* Honolulu, USA, pp. 1546-1551, May 2002.

[2] Andrew B. Watkins, A resource limited artificial immune classifier 2001. Mississippi State University.

[3] C. L. Blake and C. J. Merz. UCI Repository of Machine Learning Databases. [Online] http://www.ics.uci.edu/~mlearn/MLRepository.html . 98. University of California, Irvine, Dept. of Information and Computer Sciences.

[4] Wlodzislaw Duch. Datasets used for classification: comparison of results. [Online] http://www.phys.uni.torun.pl/kmk/projects/datasets.html . 2002. Computational Intelligence Laboratory, Department of Informatics, Nicholaus Copernicus University, Torun, Poland.

[5] Wlodzislaw Duch. Logical rules extracted from data. [Online] http://www.phys.uni.torun.pl/kmk/projects/rules.html . 2002. Computational Intelligence Laboratory, Department of Informatics, Nicholaus Copernicus University, Torun, Poland.

[6] Donald Goodman, Lois Boggess, and Andrew Watkins, "Artificial Immune System Classification of Multiple-Class Problems," *Intelligent Engineering Systems through Artificial Neural Networks: Smart Engineering System Design: Neural Networks,* pp. 179-184, 2002.

[7] Andrew Watkins, Jon Timmis, and Lois Boggess, Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm *Genetic Programming and Evolvable Machines*, vol. 5, pp. 291-317, Sep, 2004.

[8] Leandro N. de Castro and Jon Timmis. *Artificial Immune Systems: A new computational intelligence approach*, Springer-Verlag, 2002.

[9] Jerome H. Carter, The immune system as a model for classification and pattern recognition *Journal of the American Informatics Association*, vol. 7, 2000.

[10] Andrew Watkins and Jon Timmis, "Artificial Immune Recognition System (AIRS): Revisions and Refinements," *1st International Conference on Artificial Immune Systems (ICARIS2002),* University of Kent at Canterbury, pp. 173-181, 2002.

[11] Andrew B. Watkins and Lois C. Boggess, "A Resource Limited Artificial Immune Classifier," *Proceedings of Congress on Evolutionary Computation,* HI, USA, pp. 926-931, May 2002.

[12] Jon Timmis , Mark Neal, and John Hunt, An Artificial Immune System for Data Analysis *Biosystems* , vol. 55, pp. 143-150, 2000.

[13] Jon Timmis and Mark J. Neal, A Resource Limited Artificial Immune System for Data Analysis *Research and Development in Intelligent Systems XVII*, vol. pp. 19-32, Dec, 2000.

[14] Jon Timmis and Mark Neal, "Investigating the evolution and stability of a resource limited artificial immune system," *Special Workshop on Artificial Immune Systems, Gentic and Evolutionay Computtion Conference (GECCO) 2000,* Las Vegas, Nevada, U.S.A., pp. 40-41, 2000.

[15] Leandro N. de Castro and Fernando J. Von Zuben, "The Clonal Selection Algorithm with Engineering Applications," *GECCO 2000, Workshop on Artificial Immune Systems and Their Applications,* Las Vegas, USA, pp. 36-37, 2000.

[16] Leandro N. de Castro and Fernando J. Von Zuben, Learning and Optimization Using the Clonal Selection Principle *IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems*, vol. 6, pp. 239-251, 2002.

[17] Lois Boggess and Janna S. Hamaker, "The Effect of Irrelevant Features on AIRS, an Artificial Immune-Based Classifier," *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE),* pp. 219-224, 2003.

[18] Gaurav Marwah and Lois Boggess, "Artificial Immune Systems for classification: Some issues ," *First International Conference on Artificial Immune Systems,* September 2002, pp. 149-153, 2002.

[19] Janna S. Hamaker and Lois Boggess, "Non-Euclidean distance measures in AIRS, an artificial immune classification system," *Proceedings of the 2004 Congress of Evolutionary Computation,* 2004.

[20] Donald Goodman, Lois Boggess, and Andrew Watkins, "An Investigation into the Source of Power for AIRS, an Artificial Immune Classification System," *Proceedings of the International Joint Conference on Neural Networks (IJCNN'03),* pp. 1678-1683, 2003.

[21] Donald E. Goodman and Lois C. Boggess, "The role of hypothesis filtering in (AIRS), an artificial immune classifier," *Intelligent Engineering Systems Through Artificial Neural Networks,* pp. 243-248, 2004.

[22] Andrew Watkins and Jon Timmis, "Exploiting Parallelism Inherent in AIRS, an

Artificial Immune Classifier," *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS2004),* Catania, Italy, pp. 427-438, 2004.

[23] Andrew B. Watkins, Exploiting Immunological Metaphors in the Development of Serial, Parallel, and Distributed Learning Algorithms. 2004. University of Kent.

[24] Julie Greensmith, New Frontiers For An Artificial Immune System 2003. University of Leeds, Hewlett Packard Labs Technical Report number HPL.

[25] Julie Greensmith and Steve Cayzer, "An Artificial Immune Approach to Semantic Document Classification," *Proceedings of ICARIS 2003, the 2nd International Conference on Artificial Immune Systems,* 2003.

[26] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*, San Francisco: Morgan Kaufmann, 2000.

12. Additional Resources

This section contains a number of additional web resources related to the field of AIS, AIRS and the WEKA machine-learning workbench.

12.1 Homepages of researches involved in the history of AIRS

- Andrew Watkins Home Page
  - http://www.cs.kent.ac.uk/people/rpg/abw5/home.html
- Jonathan Timmis Home Page
  - http://www.cs.kent.ac.uk/people/staff/jt6/
- Donald Goodman Home Page
  - http://artificial-science.org/
- Julie Greensmith Home Page
  - http://www.cs.nott.ac.uk/~jqg
- Leandro N. de Castro Home Page
  - http://www.dca.fee.unicamp.br/~lnunes/
- Jonathan Timmis Home Page
  - http://www.cs.kent.ac.uk/people/staff/jt6/
- Mark Neal Home Page
  - http://users.aber.ac.uk/mjn/

12.2 Artificial Immune System Groups and Organisations

- International Conferences on Artificial Immune Systems
  - http://www.artificial-immune-systems.org/
- AIS at Institute of Computer Science Polish Academy of Sciences Poland
  - http://www.ipipan.waw.pl/~stw/ais/index.html
- The ISYS Project
  - http://www.aber.ac.uk/~dcswww/ISYS/
- Artificial Immune Systems and their Applications:
  - http://ais.cs.memphis.edu/home/

12.3 Weka Resources

- WEKA Home Page
  - http://www.cs.waikato.ac.nz/~ml
- WEKA Sourceforge Project
  - http://sourceforge.net/projects/weka/
- WEKA LVQ implementation (also by this author)
  - http://www.users.on.net/~nbcraven/jasonbrownlee/