A Resource Limited Artificial Immune Classifier

Andrew B. Watkins
Computing Laboratory, Univesity of Kent at Canterbury
and MPI Software Technology, Inc.
abw5@ukc.ac.uk

Lois C. Boggess
Intellgient Systems Laboratory, Department of Computer Science,
Mississippi State University.
lboggess@cs.msstate.edu

Abstract – This paper presents a new supervised learning paradigm inspired by mechanisms exhibited in immune systems. This work provides an explication of a resource limited artifical immune classification algorithm, named AIRS (Artificial Immune Recognition System), and provides results on simulated data sets to demonstrate the fundamental behavior of the algorithm.

I. Introduction

This paper presents a new supervised learning paradigm, resource limited artificial immune classifiers, inspired by mechanisms exhibited in biological and artificial immune systems. The key abstractions gleaned from these immune systems include resource competition, clonal selection, affinity maturation, and memory cell retention.

The work presented here draws inspiration from several sources within the field of Aritificial Immune Systems (AIS). This includes the key represenational concepts of B-Cells, artificial recognition balls (ARBs), and resource limitation drawn from [1], [2], and [3]. Also pivotal in the development of an immune-system inspired classifier is the use of a representation of long-lived memory cells drawn from[4] and [5]. We should also mention that to date we know of only one other attempt to use immune system principles to develop a supervised learning system [6]. However, the work presented here differs significantly for this previous work by Carter.

This paper provides an explication of a resource limited artifical immune classification algorithm, named AIRS (Artificial Immune Recognition System). Experimental results on simulated data sets demonstrate the behavior of the AIRS algorithm as a classification technique.

II. Tour of the Algorithm

This section presents a tour of the AIRS algorithm. In particular, this section presents an overview of the primary routines, methods, and equations used in the training and building of an immune-system based classifier. There are four primary stages involved in the AIRS algorithm. The first stage is data normalization and initialization. The second stage is memory cell identification and ARB generation. The third stage is competition for resources in the development of a candidate memory cell. The final stage of the training algorithm is the potential introduction of the candidate memory cell into the set of established memory cells.

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

- Let MC represent the set of memory cells and mc represent an individual member of this set.
- Let $ag.c$ represent the class of a given antigen, ag, where $ag.c \in C = \{1, 2, \ldots nc\}$ and nc is the number of classes in the data set.
- Let $mc.c$ represent the class of a given memory cell, mc, where $mc.c \in C = \{1, 2, \ldots nc\}$.
- Define $MC_c \subseteq MC = \{MC_1 \cup MC_2 \cup \ldots MC_{nc}\}$ and $mc \in MC_c$ iff $mc.c \equiv c$.
- Let $ag.f$ and $mc.f$ represent the feature vector of a given antigen and memory cell, ag and mc, respectively. Let $ag.f_i$ represent the value of the ith feature in $ag.f$ and $mc.f_i$ the value of the ith value of $mc.f$.
- Let AB represent the set of ARBs, or the population of existing cells, and MU represent a set of mutated clones of ARBs. Furthermore, let ab represent a single ARB where $ab \in AB$.
- Let $ab.c$ represent the class of a given ARB, ab, where $ab.c \in C = \{1, 2, \ldots nc\}$.
- Define $AB_c \subseteq AB = \{AB_1 \cup AB_2 \cup \ldots AB_{nc}\}$ and $ab \in AB_c$ iff $ab.c \equiv c$.
- Let $ab.stim$ represent the stimulation level of the ARB ab.
- Let $ab.resources$ represent the number of resources held by the ARB ab.
- Let $TotalNumResources$ represent the total number

0-7803-7282-4/02/$10.00 ©2002 IEEE

of system wide resources allowed.

A. Initialization

The first stage of the algorithm, initialization, can primarily be thought of as a data pre-processing stage combined with a parameter discovery stage. During initialization, first all items in the data set are normalized such that the Euclidean distance between the feature vectors of any two items is in the range of [0,1]. It is important to note that, while for the current investigation Euclidean distance is the primary metric of both affinity and stimulation, other functions could be employed as well. After normalization, the affinity threshold (AT) is calculated. For established training sets (*i.e.,* those not being generated on the fly), the affinity threshold is the average affinity value over all training data items' feature vectors. For training data supplied from a user-defined data generation function, a finite number (arbitrarily chosen at fifty for the current work) of data items are generated to be used in this calculation. The affinity threshold is calculated as described in equation (1) below:

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

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

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

B. Memory Cell Identification and ARB Generation

Once initialization is complete, training proceeds as a one-shot incremental algorithm. The first step of this stage of the algorithm is memory cell identification and ARB generation. Given a specific training antigen, ag, find the memory cell, mc_{match}, that has the following property: $mc_{match} = \arg max_{mc \in MC_{ag\_c}} stimulation(ag, mc)$, where $stimulation(x, y)$ is defined as in equation (2) below:

$$stimulation(x, y) = 1 - \text{Euclidean\_distance}(x.f, y.f) \quad (2)$$

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

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

```
MU ← ∅
MU ← MU ∪ makeARB(mc_match)
stim ← stimulation(ag, mc_match)
NumClones ← hyper_clonal_rate * clonal_rate * stim
while (| MU |< NumClones)
do
   mut ← false
   mc_clone ← mutate(mc_match, mut)
   if(mut ≡ true)
      MU ← MU ∪ makeARB(mc_clone)
done
AB ← AB ∪ MU
```

Fig. 1. Hyper-Mutation for ARB Generation

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

Fig. 2. Mutation Routine

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

C. Competition for Resources and Development of a Candidate Memory Cell

At this point a set of ARBs (AB) exists which includes mc_{match}, mutations from mc_{match}, and (possibly) remnant ARBs from responses to previously encountered antigens. Recall that the AIRS algorithm is a one-shot algorithm, so while the discussion has been divided into separate stages, each antigen goes through this entire process exactly one time. The goal of the next portion of the algorithm is to develop a candidate memory cell which is most successful in correctly classifying a given antigen, ag. This is done primarily through three mechanisms. The first mechanism is through the competition for system wide resources. Following the methods first outlined by [2] and re-examined in [3], resources are allocated to a given ARB based on its normalized stimulation value, which is used as an indication of its fitness as a recognizer of ag. The second mechanism is through the use of mutation for diversification and shape-space exploration. The third mechanism is through the use of an average stimulation threshold as a criterion for determining when to stop training on ag.

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

It is important to note here two key aspects of this resource allocation routine. First, the stimulation value of an ARB is not only determined by the stimulation function in equation 2 but is also based on the class of the ARB. The stimulation calculation method outlined in Figure 3 provides reinforcement both for those ARBs of the same class as ag that are highly stimulated by ag and for those ARBs that are of a different class as ag that do not exhibit a strong positive reaction to ag. Second, the distribution of resources is also based on the class of the ARB. This is done to provide additional reinforcement for those ARBs of the same class as ag without losing the

$minStim \leftarrow 2.0$
$maxStim \leftarrow 0.0$
foreach($ab \in AB$)
do
  $stim \leftarrow stimulation(ag, ab)$
  if ($stim < minStim$)
    $minStim \leftarrow stim$
  if ($stim > maxStim$)
    $maxStim \leftarrow stim$
  $ab.stim \leftarrow stim$
done
foreach($ab \in AB$)
do
  if($ab.c \equiv ag.c$)
    $ab.stim \leftarrow \frac{ab.stim - minStim}{maxStim - minStim}$
  else
    $ab.stim \leftarrow 1 - \frac{ab.stim - minStim}{maxStim - minStim}$
  $ab.resources \leftarrow ab.stim * clonal\_rate$
done
$i \leftarrow 1$
while($i \leq nc$)
do
  $resAlloc \leftarrow \sum_{j=1}^{|AB_i|} ab_j.resources, ab_j \in AB_i$
  if($i \equiv ag.c$)
    $NResAllowed \leftarrow \frac{TotalNumResources}{2}$
  else
    $NResAllowed \leftarrow \frac{TotalNumResources}{2*(nc-1)}$
  while($resAlloc > NResAllowed$)
  do
    $NResRemove \leftarrow resAlloc - NResAllowed$
    $ab_{remove} \leftarrow argmin_{ab \in AB_i}(ab.stim)$
    if($ab_{remove}.resources \leq NResRemove$)
      $AB_i \leftarrow AB_i - ab_{remove}$
      $resAlloc \leftarrow resAlloc - ab_{remove}.resources$
    else
      $ab_{remove}.resources \leftarrow ab_{remove}.resources - NResRemove$
      $resAlloc \leftarrow resAlloc - NResRemove$
  done
  $i \leftarrow i + 1$
done

Fig. 3. Stimulation, Resource Allocation, and ARB Removal

potentially positive qualities of the remaining ARBs for reaction to future antigens.

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

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

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

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

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

Fig. 4. Mutation of Surviving ARB

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

D. Memory Cell Introduction

The final stage in the training routine is the potential introduction of the just-developed candidate memory cell, $mc_{candidate}$, into the set of existing memory cells MC. It is during this stage that the affinity threshold calculated during initialization becomes critical as it dictates whether the $mc_{candidate}$ replaces mc_{match} that was previously identified. The candidate memory cell is added to the set of memory cells only if it is more stimulated by the training antigen, ag, than mc_{match}, where stimulation is defined as in equation (2). If this test is passed, then if the affinity between $mc_{candidate}$ and mc_{match} is less than the product of the affinity threshold and the user-defined affinity threshold scalar (ATS), then $mc_{candidate}$ replaces mc_{match} in the set of memory cells. This memory cell introduction method is presented in figure 5.

```
CandStim ← stimulation(ag, mc_candidate)
MatchStim ← stimulation(ag, mc_match)
CellAff ← affinity(mc_candidate, mc_match)
if(CandStim > MatchStim)
  if(CellAff < AT * ATS)
    MC ← MC − mc_match
  MC ← MC ∪ mc_candidate
```

Fig. 5. Memory Cell Introduction

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

0-7803-7282-4/02/$10.00 ©2002 IEEE

E. Classification

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

III. Results on Simulated Data Sets

A. Data Sets and Experimental Design

To demonstrate and investigate the behavior and principles of the current implementation of the AIRS algorithm, two somewhat basic simulated data sets are employed. Both data sets represent two-dimensional two-class problems.

The training and test data sets for these experiments are generated by randomly choosing points in the data space. For the training set, the data generation is done during run time through the use of a data generation function called from within the training routine. For the test set, fifty data points were randomly generated for use in assessing the algorithm's ability to classify previously unseen data.

The primary purpose of this set of experiments was a demonstration and investigation of some of the basic principles of the AIRS algorithm. To this effect, these experiments were kept relatively simple. That is, only two-dimensional, two-class data sets were used, only one run of the algorithm on each data set is examined, and only one set of parameter settings is explored. Each training data item was randomly generated at run time and exposed to the evolving system. After each exposure, the performance of the system on the previously generated test set was evaluated. It should be stressed here that this performance assessment in no way influenced the behavior of the system or the training on subsequent data items. In order to calculate the affinity threshold (AT) for these simulated data sets, fifty data items were randomly generated. These fifty data items were subsequently discarded after being used for the affinity threshold calculation.

B. Results

With any classification algorithm, there are two primary questions which must be addressed. First, can the algorithm develop a representation of the classes in the data set from exposure to training items? Second, can the algorithm, using this representation, generalize in order to accurately classify previously unseen data items? To address this first concern, in Figures 6 and 7 we present a visualization of the evolved memory cells after exposure to the 250 training antigens along with those 250 training items. The lines in these figures represent the original class boundaries. As can be seen in these two figures, the memory cells evolved by the AIRS algorithm have developed a credible representation of the classes in the data set. Close examination of these figures reveals that this representation is not wholly perfect. This is particularly true in areas close to the class boundaries, as would be expected. One item of interest in Figure 7 is that none of the class 1 memory cells are outside of the original class 1 boundaries, whereas there are a handful of class 0 memory cells that have bled over into class 1 territory. One possible explanation for this is that class 0 is less distinctly defined in space than class 1, which provides some ambiguity in the characteristics of a class 0 data item. Another item of note in these 2 figures is that there are far fewer than 250 memory cells that evolved from the 250 training data items. Thus, AIRS also exhibits a degree of data reduction. What is also intriguing about these two figures is not just that there are far fewer memory cells than training items, but also that it is fairly easy to discern, in places, which memory cells are representing which groups of training items.

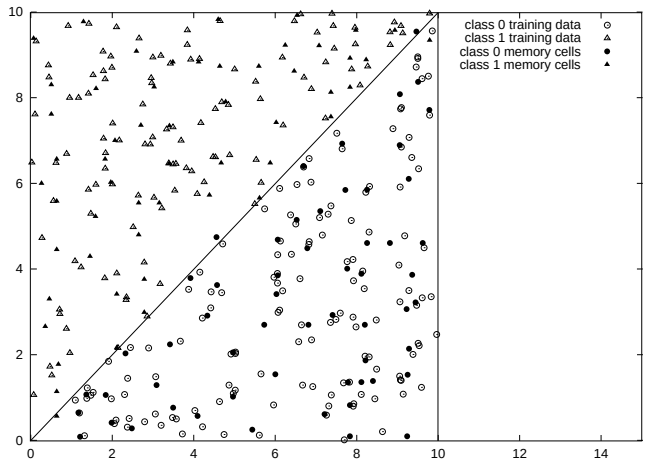

Fig. 6. Memory Cells and Training Antigens After 250 Antigens (Linearly Separable Data Set)

Another item of interest in these two figures is the outlier memory cells. While an initial supposition might be that these outliers developed in reaction to training items fairly close to the class boundary lines, examining these two figures reveals that some of the outlier memory cells are no closer to the training items than memory cells within the class boundaries. One possible explanation for this is the somewhat naïve method for memory

0-7803-7282-4/02/$10.00 ©2002 IEEE

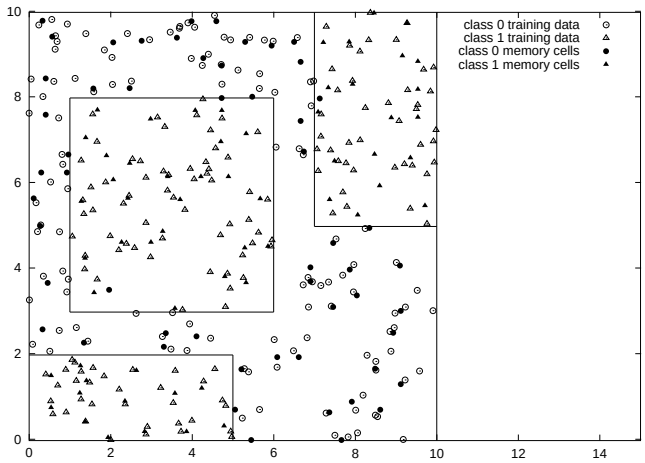

Fig. 7. Memory Cells and Training Antigens After 250 Antigens

cell replacement in the AIRS algorithm. Recall from section II that memory cell replacement only occurs when a candidate memory cell is closer to the antigen than the established match memory cell and when the candidate memory cell is within a user defined distance of the match memory cell. While this mechanism is extremely useful in removing redundant memory cells from the set of memory cells, it does not necessarily ensure that only the best classifying memory cells are retained. For the outlier cells in Figures 6 and 7 it is possible that the outlier cells developed first and that, while subsequently evolved memory cells were more stimulated by training antigens than these outlier cells, none of these later evolved memory cells were close enough to the outlier cells to warrant the removal of the outlier cells.

Now that we have provided a demonstration of the AIRS algorithm's ability to develop a representation of the classes present in these two simple data sets, we next turn to the question of classification of previously unseen data items using this representation. For these experiments the quality of classification is only assessed through overall accuracy. Recall from II.E that classification is performed in a k-nearest neighbor approach with the classification of an item being based on the majority vote among the k nearest memory cells to the item. For the current experiments k values of 1 and 3 were chosen. At the end of the introduction of 250 antigens, the system is able to accurately classify 94% and 98% (47/50 and 49/50) for $k = 1$ and $k = 3$, respectively, of the linearly separable test set and demonstrates a training set accuracy rate of 97% and 98% on the training set for these same k values. And the system is able to accurately classify 86% and 94% (43/50 and 47/50) for $k = 1$ and $k = 3$, respectively, of the non-linearly separable test set and demonstrates a training set accuracy rate of 97% and 94% for these same k values. Not surprisingly, the accuracy on the non-linearly separable data set is less than the classification accuracy on the linearly separable data set. This is to be expected for a more complex domain. Nevertheless, even for more complex data sets the AIRS algorithm is able to perform fairly well as a classifier.

IV. Conclusions

This paper has introduced a new supervised learning technique based upon immunological principles. The Artificial Immune Recognition System (AIRS) algorithm presented here demonstrates that the development of a classification system using immune-system inspired components is feasible. We have provided a detailed explication of the mechanisms of the algorithm. The key mechanisms and concepts embodied in AIRS are memory cell development, resource competition, affinity maturation, and clonal selection. We provided initial results of the behavior of AIRS on two simulated data sets. These results demonstrated that AIRS can both learn class representations in a data set and can successfully classify previously unseen data. While not presented in this paper, it should be noted that AIRS has also been shown [7] to perform as well as, or better than, other, more established, classification systems on the iris, ionosphere, pima indians diabetes, and sonar data sets availble from the UCI machine learning repository [8].

References

[1] John E. Hunt and Denise E. Cooke, "Learning using an artificial immune system," *Journal of Network and Computer Applications*, vol. 19, pp. 189–212, 1996.

[2] Jon Timmis, Mark Neal, and John Hunt, "An artificial immune system for data analysis," *Biosystems*, vol. 55, no. 1/3, pp. 143–150, 2000.

[3] Thomas Knight and Jon Timmis, "Assessing the performance of the resource limited artificial immune system AINE," Tech. Rep. 3-01, Computing Laboratory, University of Kent at Canterbury, May 2001.

[4] Leandro N. de Castro and Fernando J. Von Zuben, "aiNet: An artificial immune network for data analysis," To appear in *Data Mining: A Heuristic Approach*, Hussein A. Abbass, Ruhul A. Sarker, and Charles S. Newton (Eds.), Idea Group Publishing, USA. ftp.dca.fee.unicamp.br/pub/docs/vonzuben/lnunes/, 2001.

[5] Leandro N. de Castro and Fernando J. Von Zuben, "Learning and optimization using the clonal selection principle," To appear in *IEEE Transactions on Evolutionary Computation*, Special issue on Artificial Immune Systems. ftp.dca.fee.unicamp.br/pub/docs/vonzuben/lnunes/, 2001.

[6] Jerome H. Carter, "The immune system as a model for pattern recognition and classification," *Journal of the American Medical Informatics Association*, vol. 7, no. 1, pp. 28–41, jan/feb 2000.

[7] Andrew B. Watkins, "AIRS: A resource limited artificial immune classifier," M.S. thesis, Mississippi State University, December 2001, http://nt.library.msstate.edu/etd/show.asp?etd=etd-11052001-102048.

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