Exploiting Parallelism Inherent in AIRS, an Artificial Immune Classifier

[1] Computing Laboratory, University of Kent, UK
{abw5,jt6}@kent.ac.uk
http://www.cs.kent.ac.uk/~abw5/
[2] Department of Computer Science and Engineering, Mississippi State University, USA

Abstract. The mammalian immune system is a highly complex, inherently parallel, distributed system. The field of Artificial Immune Systems (AIS) has developed a wide variety of algorithms inspired by the immune system, few of which appear to capitalize on the parallel nature of the system from which inspiration was taken. The work in this paper presents the first steps at realizing a parallel artificial immune system for classification. A simple parallel version of the classification algorithm Artificial Immune Recognition System (AIRS) is presented. Initial results indicate that a decrease in overall runtime can be achieved through fairly naïve techniques. The need for more theoretical models of the behavior of the algorithm is discussed.

1 Introduction

Among the oft-cited reasons for exploring mammalian immune systems as a source of inspiration for computational problem solving include the observations that the immune system is inherently parallel and distributed with many diverse components working simultaneously and in cooperation to provide all of the services that the immune system provides [1,2]. Within the AIS community, there has been some exploration of the distributed nature of the immune system as evidenced in algorithms for network intrusion detection (e.g., [3,4]) as well as some ideas for distributed robot control (e.g., [5,6]), to name a couple of examples. However, very little has been done in the realm of parallel AIS–that is, applying methods to parallelize existing AIS algorithms in the hopes of efficiency (or other) gains. While just parallelizing AIS algorithms is, admittedly, venturing fairly far afield from the initial inspiration found in the immune system, the computational gains through this exercise could well be worth the (possible) side-track. Additionally, this exploration may provide some insight into other relevant areas of AIS, such as ways to incorporate diversity or even understanding the need for such.

The exploitation of parallelism inherent in many algorithms has provided definite gains in efficiency and lent insight into the limitations of the algorithms [7,8]. One example of this within the field of AIS was a very basic study of a parallel version of the CLONALG algorithm [9]. That study took advantage of the embarrassingly parallel nature of this basic AIS algorithm and demonstrated that parallel techniques can be effectively applied to AIS. This paper builds upon the lessons learned in the parallelization of CLONALG to parallelize another immune learning algorithm: AIRS. While some theoretical results are hinted, the results discussed here are very much of an empirical nature with most of the required theoretical analysis still needing to be performed.

The remainder of this paper details these initial results in parallelizing AIRS. Section 2 gives a brief overview of the serial version of the AIRS algorithm. Section 3 discusses the issues involved with parallelizing this algorithm and provides results from an initial method for this parallelization. Section 4 discusses the role of memory cells in AIRS, the impact of the initial parallel technique on the number of memory cells produced, and a possible way to overcome this apparent issue. Section 5 presents a third memory cell merging technique and the results obtained from adopting this method for solving the memory cell issue. Finally, section 6 offers some concluding remarks about this initial study of parallel AIRS.

2 Overview of the AIRS Algorithm

Developed in 2001, the Artificial Immune Recognition System (AIRS) algorithm was introduced as one of the first immune-inspired supervised learning algorithms and has subsequently gone through a period of study and refinement [10,11,12,13,14,15,16,17,18,19,20][1]. To use classifications from [1], AIRS is a bone-marrow, clonal selection type of immune-inspired algorithm, and, as with many AIS algorithms, immune-**inspired** is the key word. We do not pretend to imply that AIRS directly models any immunological process, but rather AIRS employs some components that can metaphorically relate to some immunological components. In the AIS community, AIRS has two basic precursor algorithms: CLONALG [21] and AINE [22]. AIRS resembles CLONALG in the sense that both algorithms are concerned with developing a set of memory cells that give a representation of the learned environment. AIRS also employs affinity maturation and somatic hypermutation schemes that are similar to what is found in CLONALG. From AINE, AIRS has borrowed population control mechanisms and the concept of an abstract B-cell which represents a concentration of identical B-cells (referred to as Artificial Recognition Balls in previous papers). AIRS has also adopted from AINE the use of an affinity threshold for some learning mechanisms. It should be noted that while AIRS does owe some debt of

[1] There is a debate concerning the label of supervised learning for AIRS. The authors are of the view that supervised learning is any learning system which utilizes knowledge of a training example's actual class in the building of its representation of the problem space. While AIRS does not use this information to directly minimize some error function (as seen with neural networks), it does utilize classification information about the training instances to create its world-view. Therefore, we feel that the label of supervised learning is more apt than that of reinforcement learning.

inspiration to AINE, AIRS is a population based algorithm and not a network algorithm like AINE.

While we will not detail the entire algorithm here, we do want to highlight the key parts of AIRS that will allow for understanding of the parallelization[2]. Like CLONALG, AIRS is concerned with the discovery/development of a set of memory cells that can encapsulate the training data. Basically, this is done in a two-stage process of first evolving a candidate memory cell and then determining if this candidate cell should be added to the overall pool of memory cells. This process can be outlined as follows:

1. Compare a training instance with all memory cells of the same class and find the memory cell with the best affinity for the training instance[3]. We will refer to this memory cell as mc_{match}.
2. Clone and mutate mc_{match} in proportion to its affinity to create a pool of abstract B-Cells.
3. Calculate the affinity of each B-Cell with the training instance.
4. Allocate resources to each B-Cell based on its affinity.
5. Remove the weakest B-Cells until the number of resources returns to a preset limit.
6. If the average affinity of the surviving B-Cells is above a certain level, continue to step 7. Else, clone and mutate these surviving B-Cells based on their affinity and return to step 3.
7. Choose the best B-Cell as a candidate memory cell (mc_{cand}).
8. If the affinity of mc_{cand} for the training instance is better than the affinity of mc_{match}, then add mc_{cand} to the memory cell pool. If, in addition to this, the affinity between mc_{cand} and mc_{match} is within a certain threshold, then remove mc_{match} from the memory cell pool.
9. Repeat from step 1 until all training instances have been presented.

Once this training routine is complete, AIRS classifies instances using k-nearest neighbor with the developed set of memory cells.

3 Parallelizing AIRS

Having reviewed the serial version of AIRS, we turn our attention to our initial strategies for parallelizing this algorithm. Our primary motivation for these experiments is computational efficiency. We would like to employ mechanisms of harnessing the power of multiple processors applied to the same learning task rather than relying solely on a single processor. This ability will, in theory, allow us to apply AIRS to problem sets of a larger scale without sacrificing some of the appealing features of the algorithm. Secondary goals of this work include gaining more insight into the processes necessary to parallelize immune algorithms, in

[2] See [14] for the pseudocode of AIRS.

[3] Affinity is currently defined as Euclidean distance. We are looking for the closest memory cell of the same class as the training instance.

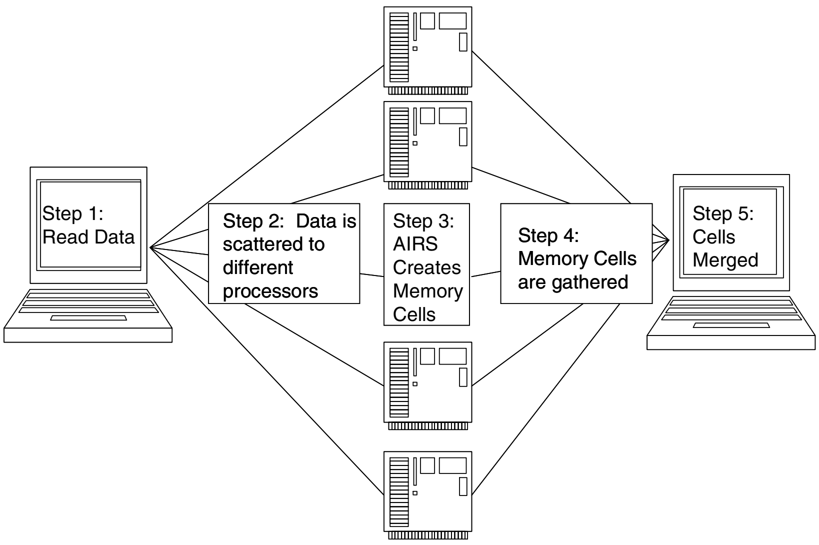
Step 1:
Read Data
Step 2: Data is scattered to different processors
Step 3: AIRS Creates Memory Cells
Step 4: Memory Cells are gathered
Step 5: Cells Merged

Fig. 1. Overview of Parallel AIRS

general, as well as the implication of such work for the study of the role of diversity and distributedness in AIS.

Our initial approach to parallelizing this process is the same as the approach to parallelizing CLONALG presented in [9]: we partition the training data into *np* (number of processes) pieces and allow each of the processors to train on the separate portions of the training data. Figure 1 depicts this process. Unfortunately, unlike CLONALG which simply evolves one memory cell for each training data item, AIRS actually employs some degree of interaction between the candidate cells and the previously established memory cells. Partitioning the training data and allowing multiple copies of AIRS to run on these fractions of the data in essence creates *np* separate memory cell pools. It introduces a (possibly) significant difference in behavior from the serial version. So, when studying this parallelism, we must examine not only the computational efficiency we gain through this use of multiple processors, but we must also learn how evolving these memory cell pools in isolation of one another effects the overall performance of the algorithm.

Algorithmically, based on what is described in section 2, the parallel version behaves in the following manner:

1. Read in the training data at the root process.
2. Scatter the training data to the *np* processes.
3. Execute, on each process, steps 1 through 9 from the serial version of the algorithm on the portion of the training data obtained.
4. Gather the developed memory cells from each processes back to the root.
5. Merge the gathered memory cells into a single memory cell pool for classification.

Table 1. Iris Results: Concatenation

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	95.600%(3.591)	64.380(4.295)	0.346(0.010)	1.000
2	95.800%(3.973)	74.920(4.642)	0.393(0.084)	0.441
4	96.133%(2.886)	86.960(4.785)	0.282(0.092)	0.307
8	95.400%(3.222)	97.280(5.474)	0.282(0.057)	0.153
16	95.333%(3.869)	104.920(5.279)	0.372(0.083)	0.058
24	95.533%(3.665)	108.300(4.879)	0.499(0.065)	0.029

Table 2. Pima Diabetes Results: Concatenation

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	73.084%(4.564)	277.050(9.314)	3.540(0.091)	1.000
2	73.347%(5.200)	316.960(13.153)	2.668(0.120)	0.663
4	73.362%(5.186)	359.470(15.344)	1.848(0.104)	0.479
8	74.115%(4.866)	402.270(22.718)	1.557(0.057)	0.284
16	74.494%(4.802)	448.270(26.608)	1.362(0.057)	0.162
24	74.451%(4.605)	475.590(31.687)	1.330(0.082)	0.111

Table 3. Sonar Results: Concatenation

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	85.144%(8.097)	172.885(4.074)	57.141(3.582)	1.000
2	84.615%(8.946)	179.038(3.463)	34.738(3.277)	0.822
4	83.894%(8.341)	183.546(2.883)	20.189(1.539)	0.708
8	85.288%(8.731)	186.677(2.589)	12.141(1.261)	0.588
16	84.904%(9.396)	189.038(1.914)	7.255(0.867)	0.492
24	84.567%(8.602)	189.838(1.656)	5.769(0.873)	0.413

Since this method of parallelism creates *np* separate memory cell pools and since our classification is performed using a single memory cell pool, we must devise a method for merging the separate memory cell pools into one pool.

Initially, we simply gathered each of the *np* memory cell pools at the root processor and concatenated these into a single large memory cell pool. While this is an extremely naïve approach, as tables 1, 2, and 3 demonstrate, we were still able to achieve overall speedup in the process[4]. On a technical note, for the experiments presented in this paper, we used the Iris, Pima Diabetes, and Sonar data sets that were used in previous studies of AIRS [14]. For all of these we took an average over 10 cross-validated runs and tested the parallel version on an increasing number of processors. A cluster of dual-processor 2.4Ghz Xeons were used. The Message Passing Interface (MPI)[23] was used as the communication library and communication took place over a Gigabit Ethernet network.

There are a couple of observations to be made from this initial set of experiments. Foremost, for our current purposes, there is a gain in overall

[4] Values in parentheses represent standard deviation.

runtime of the algorithm by parallelizing it, and this speedup is achieved without any loss in classification accuracy. The iris data set is somewhat anomalous to this general observation The results indicate that for this data set some amount of gain can be had by utilizing more processing power; however, at some point this gain is no longer achieved. This point is probably where the communication and setup time involved in this type of parallelization outstrips the usefulness of having more processors evaluating the training data. Being able to predict what this point is in general will be the focus of our future theoretical/analytical work as we explore this parallel version of AIRS more thoroughly. However, looking at the parallel efficiency, there appears to be something subtler occurring than just a simple speedup from more processing power.

Parallel efficiency can be defined as:

$$E(P) = \frac{T(1)}{P * T(P)}$$

where P is the number or processors, $T(1)$ is the time for the serial version of the algorithms, and $T(P)$ is the time for the parallel version of the algorithm to run on P processors. Ideally, we would have a parallel efficiency of 1. However, this can rarely be achieved due to issues such as communication time and setup. For AIRS, we might initially assume that the more feature vectors in the training set, the greater the parallel efficiency. However, this is not the case. The pima diabetes data set has 691 training items in it whereas the sonar data set has only 192 data items in the training set. Yet, examining the parallel efficiency results for these two data sets reveals that the sonar data set has much more to gain from parallelization than does the pima diabetes data. The explanation for this seeming discrepancy is in the number of features in each feature vector. The pima diabetes data has only eight features per feature vector, whereas the sonar data has 60 (iris has 4 features, incidentally). That our overall runtime (and its parallel efficiency) is predicated on the number of features in the data set should not be surprising. As with parallel GAs [7], the parallel version of AIRS is essentially dividing up the work of fitness (or affinity) evaluations. For the current version of AIRS, affinity is determined based on Euclidean distance which is a metric whose evaluation grows linearly with the number of features. Thus, more gain will be seen from data sets with both a large number of features and a large number of training instances when applying the parallel version of AIRS.

4 Memory Cells

The results in the previous section exhibited another side-effect of note: with parallelization comes an increase in memory cells. One of the hallmarks of AIRS has been its data reduction capabilities. As presented in [14], AIRS has been shown to reduce the amount of data needed to classify a given data set up to 75%. This data reduction is measured in the number of memory cells present in the final classifier. To get an empirical sense of how the size of the memory cell set effects the classification time, we ran a set of experiments in which we

Table 4. Comparison of Runtimes for KNN and AIRS

Tr	Test	MC	T_{test}(KNN)	T_{test}(AIRS)	T(KNN)	T(AIRS)
692	77	277.850	0.149	0.072	1.290	3.521
615	153	254.040	0.276	0.115	1.181	3.048
512	256	217.433	0.373	0.154	1.012	2.468

compared AIRS to k-nearest neighbor (k-nn). Recall, that basic k-nn simply takes all of the training instances as examples and then classifies the test set through a majority voting scheme. AIRS first grows a set of memory cells which are then used to classify the test set. The results from these basic experiments on the pima diabetes data set are given in table 4 and provides a comparison between the number of training and test cases used, the number of memory cells developed by AIRS, and the difference in testing and overall runtime for the k-nn and AIRS. Not surprisingly, when AIRS has greatly reduced the data set, there is a speed up in time to classify the test set[5].

Since the classification speed of AIRS is based on the number of memory cells in the final pool, it is important to understand what impact parallelizing AIRS would have on the size of this set. In the serial version of AIRS, the minimum number of memory cells allowed is the number of classes that exist in the data set (one memory cell per class), and the maximum number is the number of data items in the training set, n, (one memory cell per training vector). For the parallel version, assuming that each process has examples of each class, the minimum at each process is the number of classes (nc); whereas, the maximum would be n/np. So, in the concatenation version of merging, the minimum number of memory cells in the final classifier increases from nc to $nc * np$. While one might suppose that the number of memory cells obtained through either the serial or parallel versions should be the same, it should be remembered that step 1 and (by implication) step 8 of the serial version depend on interaction with the entire memory cell pool. This interaction is not available in the current parallel version.

Our second approach to the merging stage is an attempt to minimize the number of memory cells that resulted from the pure concatenation approach. This method uses an affinity-based technique similar to step 8 in the serial version to reduce the size of the final memory cell pool. After gathering all the memory cells to the root process, they were then separated by class. Within each class grouping, a pairwise calculation of affinity between the memory cells was performed. If the affinity between two memory cells was less than the affinity threshold multiplied by the affinity threshold scalar, then only one of the memory cells was maintained in the final pool. That is, if this relation:

$$\text{affinity}(mc_i, mc_j) < AT * ATS \tag{1}$$

[5] In all fairness, it should be mentioned that the time to train in k-nn is virtually nothing, whereas the time to train in AIRS can be significant (when compared to 0). However, once the classifier is trained, it is the classification time that becomes most important as this is the task for which the classifier has been trained.

Table 5. Iris Results: Affinity-Based Merging

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	95.867%(3.535)	62.520(4.687)	0.379(0.218)	1.000
2	95.200%(3.698)	68.020(3.771)	0.379(0.047)	0.500
4	95.267%(3.815)	72.740(5.620)	0.264(0.067)	0.359
8	95.333%(3.159)	79.600(7.025)	0.279(0.072)	0.170
16	95.267%(4.046)	84.720(9.630)	0.381(0.070)	0.062
24	94.867%(3.822)	88.380(13.351)	0.514(0.090)	0.031

Table 6. Pima Diabetes Results: Affinity-Based Merging

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	73.321%(4.908)	276.090(10.363)	3.739(0.086)	1.000
2	73.806%(4.921)	305.350(14.519)	2.944(0.177)	0.635
4	73.504%(4.444)	340.660(13.444)	2.160(0.107)	0.433
8	73.766%(4.731)	373.150(23.934)	1.891(0.100)	0.247
16	74.280%(4.585)	412.600(31.110)	1.776(0.140)	0.132
24	73.961%(4.811)	429.570(41.682)	1.768(0.174)	0.088

Table 7. Sonar Results: Affinity-Based Merging

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	84.808%(8.523)	172.585(4.124)	58.320(3.057)	1.000
2	83.846%(8.934)	179.346(3.258)	35.291(4.128)	0.826
4	85.625%(8.895)	184.008(2.610)	20.407(1.863)	0.714
8	84.712%(8.668)	186.754(2.392)	12.419(1.157)	0.587
16	85.000%(8.403)	188.992(1.882)	7.489(0.692)	0.487
24	84.375%(9.282)	189.862(1.529)	5.996(0.837)	0.405

(where mc_i and mc_j are two memory cells of the same class, the affinity threshold (AT) had been calculated across all of the training antigens as shown in equation 2, and the affinity threshold scalar (ATS) is set by the user) holds true, then mc_j is removed from the memory cell pool.

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

This merging technique was an initial attempt to compensate for the lack of global interaction the parallelizing process introduced. Tables 5, 6, and 7 give results when using this affinity-based merging.

5 Affinity-Based Merging Revisited

As seen in Section 4, the basic affinity-based merging technique employed did not significantly reduce the increase in memory cells present in the final classifier.

Clearly, the serial version of AIRS does not need as many memory cells to classify as accurately, so we would like to find a way to capture this further reduction in data while still employing our parallel techniques. Examining the increase in memory cells, there appears to be a roughly logarithmic increase with respect to an increase in the number of processors used. One method of remedying this increase in memory cells would be to alter the memory cell replacement criterion used in the affinity-based merging scheme by a logarithmic factor of the number of processors. That is, the criterion for removing a given memory cell is no longer as specified in equation 1, but now the following relation must hold true for the removal of a memory cell:

$$\text{affinity}(mc_i, mc_j) < AT * ATS + \text{factor} \tag{3}$$

and factor is defined as:

$$\text{factor} = AT * ATS * \text{dampener} * log(np) \tag{4}$$

With the “dampener” referred to in equation 4 being a number between 0 and 1, this change to the merging scheme relaxes the criterion for memory cell removal in the affinity-based merging scheme by a small fraction in logarithmic proportion of the number of processors used[6]. Tables 8, 9, and 10 below present results when employing this logarithmic factor to the criterion used in the affinity-based merging scheme[7].

Table 8. Iris Results: Processor Dependent, Affinity-Based Merging

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	95.533%(3.726)	63.200(4.607)	0.356(0.012)	1.000
2	95.933%(3.943)	63.720(4.953)	0.358(0.069)	0.497
4	95.467%(3.913)	64.220(4.679)	0.229(0.123)	0.388
8	95.467%(3.913)	63.520(6.234)	0.387(0.126)	0.115
16	95.067%(3.450)	64.440(10.643)	0.495(0.098)	0.045
24	95.333%(3.434)	63.160(13.872)	0.502(0.092)	0.030

Table 9. Pima Diabetes Results: Processor Dependent, Affinity-Based Merging

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	73.356%(4.827)	274.430(9.580)	3.734(0.172)	1.000
2	73.359%(4.731)	283.170(12.588)	2.888(0.159)	0.647
4	74.113%(4.524)	283.380(17.528)	1.934(0.141)	0.483
8	73.935%(5.162)	274.890(18.706)	1.589(0.081)	0.294
16	74.066%(4.794)	263.640(31.849)	1.297(0.126)	0.180
24	72.984%(5.433)	261.800(43.940)	1.203(0.165)	0.129

[6] Obviously, the scheme presented in section 4 is just a variation on this new formulation with a “dampener” value of 0.

[7] An arbitrary value of 0.1 was used for the “dampener” value for these experiments.

Table 10. Sonar Results: Processor Dependent, Affinity-Based Merging

NP	Test Set Accuracy	Memory Cells	Overall Runtime(s)	Parallel Efficiency
1	83.654%(8.954)	172.692(4.112)	58.499(3.138)	1.000
2	84.519%(9.633)	175.877(3.783)	35.231(3.286)	0.830
4	84.135%(9.213)	179.600(2.863)	20.230(1.310)	0.723
8	84.760%(9.208)	182.292(2.868)	12.386(1.421)	0.590
16	84.087%(8.808)	183.523(3.211)	7.413(0.676)	0.493
24	85.529%(9.209)	185.108(3.071)	5.806(0.746)	0.420

The results from this new merging scheme are somewhat inconclusive. Looking at the iris results (table 8), we appear to have achieved our goal of maintaining the number of memory cells in the parallel classifier at a similar level to the serial version. (As a side-note, we again see the same timing behavior with this data set that we mentioned in section 3.) However, the other two sets of results are not as obvious. While the experiments on the pima diabetes set (table 9) do exhibit a reduction in the number of memory cells in the final classifier, it is unclear if this reduction would have continued unbounded if we had tested on more and more processors. Eventually, with a significant decrease in memory cells, classification accuracy would decrease as well. And, with the sonar data set experiments (table 10), our new scheme appears to have had virtually no impact on the rate of growth of the number of memory cells. What all of this indicates may be simply that we have introduced another parameter (the “dampener” in equation 4) and that we need to determine the appropriate setting for this parameter for each classification task at hand.

6 Conclusion

Our goal with this study was to explore ways of exploiting parallelism inherent in an artificial immune system for decreased overall runtime. Using a very basic mechanism for this parallelism, we have shown that there are definite benefits (computationally, at least) from this exploration. However, more questions were raised than were answered here. Ideally, we would like a way to predict the number of processors to employ to provide the most benefit. In other words, there is the need for a run-time prediction model based on input size as well as feature size.

One side-effect of our parallelization of AIRS was that its final predictive model increased in size. We explored mechanisms for reducing this size to something more comparable with the serial version. While our technique for tackling this might ultimately be the correct approach, currently its use provides inconclusive results (at best). One logical place to look for other ways of solving this problem would be the immune system itself. The immune system is inherently distributed, yet the number of memory cells in the system remains fairly constant. Examining the mechanisms used for this in nature might lend insight into how to address this problem in parallel AIRS.

In addition to some of the algorithmic and theoretical questions that need to be answered, we would also like to expand the use of AIRS and parallel AIRS to more application domains. Any good learning technique attempts to exploit domain knowledge. Given this, we also want to find ways to incorporate domain knowledge into our current learning model.

References

1. de Castro, L., Timmis, J.: Artificial immune systems: A new computational approach. Springer-Verlag, London. UK. (2002)
2. Dasgupta, D., ed.: Artificial Immune Systems and Their Applications. Springer, Berlin (1998)
3. Hofmeyr, S., Forrest, S.: Arichitecture for an aritifcial immune system. Evolutionary Computation **7(1)** (2000) 45–68
4. Kim, J.W.: Integrating Artificial Immune Algorithms for Intrusion Detection. PhD thesis, Department of Computer Science, University College London (2002)
5. Lee, D.W., Jun, H.B., Sim, K.B.: Artificial immune system for realisation of co-operative strategies and group behaviour in collective autonomous mobile robots. In: Proceedings of Fourth International Symposium on Artificial Life and Robotics, AAAI (1999) 232–235
6. Lau, H.Y., Wong, V.W.: Immunologic control framework for automated material handling. In Timmis, J., Bentley, P., Hart, E., eds.: Proceedings of the 2nd International Conference on Artificial Immune Systems. Number 2787 in Lecture Notes in Computer Science, Springer-Verlag (2003) 57–68
7. Cantú-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Acadeimic Publishers (2000)
8. Chattratichat, J., Darlington, J., Ghanem, M., Guo, Y., Hunning, H., Kohler, M., Sutiwaraphun, J., Wing To, H., Yang, D.: Large scale data mining: Challenges and responses. In: KDD-97. (1997) 143–146
9. Watkins, A., Bi, X., Phadke, A.: Parallelizing an immune-inspired algorithm for efficient pattern recognition. In Dagli, C., Buczak, A., Ghosh, J., Embrechts, M., Ersoy, O., eds.: Intelligent Engineering Systems through Artificial Neural Networks: Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life. Volume 13. ASME Press, New York (2003) 225–230
10. Watkins, A.: AIRS: A resource limited artificial immune system. Master's thesis, Mississippi State University (2001)
11. Watkins, A., Boggess, L.: A new classifier based on resource limited artificial immune systems. In: Proceedings of Congress on Evolutionary Computation, Part of the 2002 IEEE World Congress on Computational Intelligence held in Honolulu, HI, USA, May 12-17, 2002, IEEE (2002) 1546–1551
12. Watkins, A., Boggess, L.: A resource limited artificial immune classifier. In: Proceedings of Congress on Evolutionary Computation, Part of the 2002 IEEE World Congress on Computational Intelligence held in Honolulu, HI, USA, May 12-17, 2002, IEEE (2002) 926–931
13. Watkins, A., Timmis, J.: Artificial immune recognition system (AIRS): Revisions and refinements. In: Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS). (2002)

14. Watkins, A., Timmis, J., Boggess, L.: Artificial immune recognition system (AIRS): An immune inspired supervised machine learning algorithm. Genetic Programming and Evolvable Machines **5** (2004) 291–317
15. Marwah, G., Boggess, L.: Artificial immune systems for classification: Some issues. In: Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS). (2002)
16. Goodman, D., Boggess, L., Watkins, A.: Artificial immune system classification of multiple-class problems. In Dagli, C.H., Buczak, A.L., Ghosh, J., Embrechts, M.J., Ersoy, O., Kercel, S.W., eds.: Intelligent Engineering Systems Through Artificial Nerual Networks: Smart Engineering System Design: Neural Netwokrs, Fuzzy Logic, Evolutionary Programming, Data Mining, and Complex Systems. Volume 12. ASME Press, New York (2002) 179–184
17. Goodman, D., Boggess, L., Watkins, A.: An investigation into the source of power for AIRS, an artificial immune classification system. In: Proceedings of the International Joint Conference on Neural Networks 2003, Portland, OR, USA, The International Neural Network Society and the IEEE Neural Networks Society (2003) 1678–1683
18. Goodman, D., Boggess, L.: The role of hypothesis filter in AIRS, an artificial immune classifier. In Dagli, C., Buczak, A., Ghosh, J., Embrechts, M., Ersoy, O., eds.: Intelligent Engineering Systems through Artificial Neural Networks: Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life. Volume 13. ASME Press (2003) 243–248
19. Greensmith, J., Cayzer, S.: An artificial immune system approach to semantic document classification. In Timmis, J., Bentley, P., Hart, E., eds.: Proceedings of the 2nd International Conference on Artificial Immune Systems. Number 2787 in Lecture Notes in Computer Science, Springer-Verlag (2003) 136–146
20. Hamaker, J., Boggess, L.: Non-euclidean distance measures in AIRS, an artificial immune classification system. In: Proceedings of the 2004 Congress on Evolutionary Computing. (2004)
21. de Castro, L.N., von Zuben, F.: Learning and optimization using the clonal selction principle. IEEE Transactions on Evolutionary Computation **6** (2002) 239–251
22. Timmis, J., Neal, M.: A Resource Limited Artificial Immune System. Knowledge Based Systems **14** (2001) 121–130
23. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message Passing Interface. 2nd edn. MIT Press (1999)