A New Approach to Optimized Negative Selection

Brian Schmidt
Computer Science Department
College of Engineering and Applied Sciences
Western Michigan University
Kalamazoo, Michigan, USA
brian.h.schmidt@wmich.edu

Ala Al-Fuqaha
Computer Science Department
College of Engineering and Applied Sciences
Western Michigan University
Kalamazoo, Michigan, USA
ala.al-fuqaha@wmich.edu

Abstract-- **The Negative Selection (NS) algorithm is the first algorithm to come from the study of natural immune systems, and is the most widely known and applied algorithm in the field. It has been used to build intrusion detection systems along with many other security-related tasks. However, it has not been possible to use the Negative Selection algorithm on many real-world scenarios. The present research shows an optimization of the negative selection algorithm to make its execution faster. The optimized algorithm remains functionally the same, providing the same results as the unoptimized algorithm. Details are given about the optimization scheme used and the optimized negative selection algorithm is tested on the UCI Breast Cancer data set. The performance of the unoptimized negative selection algorithm is compared to the performance of the algorithm with the proposed optimization. Three claims about the function of the optimized negative selection algorithm are made and tested with four experiments. The results of the experiments are used to demonstrate that the algorithm is faster and does not change the negative selection algorithm or lower its accuracy. Although there has been research into the optimization of the Negative Selection algorithm, this work will only apply to hyper-sphere detectors, which has not been done before.**

Keywords— artificial immune system; negative selection; optimization; optimized negative selection

I. INTRODUCTION

The negative selection algorithm has been around for more than two decades. In this time, it has been adapted to many problem domains, and has found some success. Its ability to model a binary decision boundary using only samples from one class is especially useful in some circumstances. Since its inspiration is the immune system of mammals it has been widely used in anomaly detection systems and network security systems.

However, its performance has been very slow in most real-world problems. For example, in [1], Kim and Bentley explore the use of negative selection and clonal selection in an Artificial Immune System classifier, using both network data and several data sets from the UCI repository. When used with network data, their algorithm proved to be infeasible, taking too long to train. It was estimated that for an 80% detection rate it would take 1,429 years to generate the 6×108 detectors needed (Kim and Bentley, 1999). This research is used as an example of the challenges faced by the NS algorithm, however, there are several optimizations that have been developed since this publication that could be applied to it.

The focus of this paper will be to optimize both the training and classification portions of the negative selection algorithm, to make the algorithm practical in real-world problems.

The rest of the paper is organized in this manner: in Section 2 natural immune systems and Artificial Immune System classifiers are presented along with a literature review of all previous work in this area of research. Section 3 presents the optimizations proposed in this research, along with pseudo code to illustrate them. Sections 4 describes the data set and code used to test the optimization along with details of the experiments performed. Section 5 gives the graphs with the results of the experiments. Lastly, in Section 6 we draw conclusions and show a few ways in which the research could be continued.

II. ARTIFICIAL IMMUNE SYSTEMS

A. Natural Immune Systems

Since this work deals with an Artificial Immune System (AIS) algorithm, this section will provide a short introduction to the original inspiration for these algorithms.

Natural immune systems are an important part of vertebrate organisms and are in charge of protecting living beings from outside dangers. The immune systems perform two essential exercises activities: recognition of pathogens and the removal of pathogens, which is anything that can be a threat to the host organism. Natural immune systems perform these tasks through two subsystems: the innate immune system and the acquired immune system. The innate immune system is static and does not adapt to the conditions of the environment, and the acquired immune system is able to learn to recognize new pathogens. The acquired immune system is of interest to computer scientists because of its adaptability and flexibility.

The process of negative selection is accomplished through the random generation of detectors known as "antibodies". Each antibody is a simple molecule that is able to attach itself to one particular protein, recognizing it. The proteins recognized are on the surface of the organism's own cells and the surfaces of pathogen cells. The challenge is to generate antibodies that will only attach themselves to pathogens, while ignoring the organism's own cells.

The negative selection process proceeds through the creation of cells that generate antibodies randomly. However, before the cells are fully mature, they are verified in the thymus, an organ located behind the sternum in humans. The thymus tests each immature antibody-generating cell against the body's own cells, removing all cells that generate antibodies that recognize the body's own cells as pathogens.

Once an antibody attaches itself to a pathogen, it is marked for destruction and disposal. The classification rule is simply this: if one or more antibodies recognizes a tissue, then it is non-self and dangerous; if no antibody recognizes a tissue, then it is self and safe. A special attribute of the negative selection algorithm is that it only requires samples of the self tissue to perform classification.

B. The Negative Selection Algorithm

From the natural process known as negative selection, an algorithm has been developed that is able to perform binary classification. The Negative Selection algorithm is described in this section. The Negative Selection algorithm was first presented by Forrest, et al. in [2].

The training algorithm derived from the actions of the thymus is very simple. The strategy is directly applicable to data that is separated into two classes only, in the same way that the immune system recognizes self and non-self. This means that one class is chosen to be self and the other class becomes non-self. An antibody is implemented in a computer as a simple classifier called a "detector", which takes a sample from the data set and returns true or false, signaling whether or not it has recognized the sample. It is not necessary for a detector to be of any type or structure, as long as it is able to classify a sample. Another component of the algorithm is mechanism for detectors to be randomly generated.

The negative selection takes a parameter that determines the desired size of the set of detectors. Larger sets are not necessarily better, since they take more time to create and to classify samples. The algorithm proceeds iteratively, generating a random detector and testing it against the set of samples in the data set that are in the class that has been decided to be the "self" class. If the new random detector does not match any samples in the self class, then is added to the set; if it does match a sample, then it is discarded. A pseudo code listing of the negative selection training algorithm is found in Figure 1.

As can be seen in the pseudo code listing, the portion of code that is executed most often in the algorithm is the match() function, which compares one detector against one sample from the data set. This function is used many times during the execution of the algorithm. This is a good place to start when looking for ways to optimize the algorithm.

Like the classification performed by the natural immune system, the Negative Selection algorithm classification step is very simple. The classification step uses the set of detectors created by the training step. Classification is performed using the whole set of detectors, by comparing each detector to the sample to be classified. If one detector matches the sample, then it is classified as "non-self"; if no detectors match the sample, then it is classified as "self". As explained above, the choice of self and non-self classes is taken during the training step of the algorithm. Pseudo code for the classification portion of the Negative Selection Algorithm can be seen in Figure 2.

C. *Affinity Measures*

The "affinity" of a detector is defined in relation to a data sample, in the same way as in the natural immune system. The affinity of a detector-sample pair is a measurement of how well the detector matches the sample. There is no universal way to define affinity, the affinity measure is defined according to the type of data that the Negative Selection algorithm is dealing with.

Figure 1. Pseudo Code for the Negative Selection Training Algorithm

```
self: set of seen self samples
detectors: the set of detectors
matches(): a function that returns true if an antibody
recognizes an element
generate_random_antibody(): a function to generate a
random antibody

detectors = {}
WHILE !stopping_criteria
    new_antibody = generate_random_antibody()
    match = FALSE
    FOREACH { s | s ∈ self }
        IF matches(s, new_antibody)
            match = TRUE
        ENDIF
    END FOREACH
    IF !match
        detectors = detectors ∪ new_antibody
    ENDIF
ENDWHLE
```

For real-valued data, distance metrics are a popular way to define affinity. For example: Euclidian distance, Manhattan distance, Cosine distance, and Chebyshev distance. For string data, other distance metrics are available that can be used as an affinity measure. For example, Hamming distance, binary distance, edit distance, and value difference metrics. A good overview of these and other ways to calculate affinity was done by Dasgupta and Nino and can be found in [3].

D. *Detector Types*

As previously stated, there are many ways to implement detectors. However, all of the detector implementations have one thing in common: they are all simple classifiers that return a binary value. This section discusses some common implementations of detectors.

R-chunk and r-contiguous detectors work with string data. Each string is made up of a defined set of symbols. The detectors themselves are strings made of the same symbols. The r-contiguous matching rule matches a detector and a string of the same length if r contiguous symbols in both strings are identical, where r is an integer. R-chunk matching is similar to r-contiguous matching, but the detector and data are not necessarily of the same length. In [4], Dasgupta, and Majumdar use r-contiguous bit detectors for anomaly detection and classification with personnel data. In [5], Balthrop et al. use r-contiguous bit detectors for network intrusion detection. Lastly, in [6], Ayara et al. use r-contiguous bit detectors for error detection and classification in ATM machines. R-contiguous detectors are first mentioned Percus et al. in [7], and r-chunk detectors are first mentioned by in Balthrop et al. in [5].

Using the notion of affinity and distance functions mentioned in a previous section, it is possible to define hyper-sphere detectors. A hyper-sphere is defined by a center point and a radius. The detector then "matches" a point within the feature space if it falls within its hyper-sphere. This type of detector only works in feature spaces that only contain continuous (real-valued) features. The most common distance measure used is Euclidian distance. In [8], Cserey et al. use hyper-sphere detectors for image recognition, while in [9], Şahan et al. uses hyper-sphere detectors for medical diagnosis.

E. *Previous Work in Optimized AIS Negative Selection Algorithms*

The idea of creating a more efficient Negative Selection algorithm is first presented by Elberfeld and Textor in [10], in which they show how r-contiguous and r-chunk based set of detectors could be trained and used in recognition in a more efficient manner. Through the use of a specialized technique, the authors are able to compress the set of detectors.

The compression scheme used is simply using a single pattern to describe a set of several detectors. By using the patterns instead

Figure 2. Pseudo Code for the Negative Selection Classification Algorithm

```
detectors: set of detectors created by training algorithm
sample: sample to be classified
matches(): a function that returns true if an antibody
recognizes a sample

FOREACH { a | a ∈ detectors }
    IF matches(a, sample}
        return "non-self"
    ENDIF
ENFOREACH
return "self"
```

of the detectors themselves in the matching steps of the training and classification algorithms, the time complexity is lowered. The worst case time complexity of the original algorithm is exponential, but it becomes polynomial with the new technique. The results of this paper show that storing all of the detectors is unnecessary, and by compressing the set a substantial speedup is achieved.

In a similar publication [11] Liśkiewicz and Textor, also deal with the optimization of the Negative Selection algorithm. However, the technique is very different and does not use compression of the detector set at all. The authors show that it is possible to implement the Negative Selection algorithm without creating a detector set at all, depending on the type of detectors used. The paper is very thorough and provides proofs for all claims, but does not implement and test the proposed technique. The results are similar to [10], with the time complexity of the Negative Selection algorithm being reduced from exponential to polynomial. This research is, like other research by the same authors, only applicable to detectors that work with strings, like r-chunk and r-contiguous detectors.

Following [10] and [11], the ideas in these papers were combined in [12] by Elberfeld and Textor which sought to settle the question of whether the Negative Selection algorithm could be used in real-life problems. This paper presents two algorithms, both used for training an automaton, which is then used for classification. The training algorithms can be executed in polynomial time, and the automaton used for classification works in linear time. The algorithms work by simulating the Negative Selection algorithm, with the same behavior being exactly replicated by the more efficient algorithms. The algorithms use prefix trees to speed up the work, instead of the patterns used in [10].

The latest work by Textor on the subject of optimizing the execution of Negative Selection algorithms is [13]. One further variation from the previous approaches is explored in this publication, in which detectors are generated by sampling from the set of S-consistent detectors. S-consistent detectors are defined to be the set of detectors that do not match any element in the S set, which is the set of self samples. Whereas the Negative Selection Algorithm (NSA) algorithm normally samples the space of possible detectors uniformly, the author shows that by doing this differently, it is possible to speed up the execution of NSAs. This paper theoretically proves that it possible to speed up the execution of NSA by using probabilistic sampling techniques such as Markov Chain Monte Carlo methods.

In [14] and [15], Wang, Yibo, and Dong propose a faster training and classification methods for Negative Selection algorithms. The technique they show uses "neighborhoods" in the feature space to represent both detectors and samples to be classified. They also introduce a method to improve the matching operation between detectors and samples that improves the performance, especially in high dimensions.

In [16] Yang et al. the authors show a similar approach to the one used in [14] and [15]. The algorithm is called GF-RNSA, a Negative Selection algorithm which uses the concept of grid cells to speed up execution. The feature space is separated into grid cells, and detectors are generated separately for each cell, only comparing the candidate detectors with the self samples that are in the same grid cell, instead of the whole set. This approach also manages to eliminate the exponential time complexity of the NSA. This publication also showed experimental results. In [17] Wen et.al. apply a very similar technique as [16].

Lastly, a novel detector generation method is proposed in [18]. The method was developed by Ji and Dasgupta and is called V-detector. The strategy involves the statistical analysis of the data in order to improve the amount of non-self space that is covered while also minimizing the number of detectors needed to do so. The detector generation process also takes into account the boundary of the classes in the data set to improve the quality of the set of detectors. The detectors are allowed to be of variable size, as well. These techniques allow the algorithm to be very efficient. The scheme is also applicable across many different detector types.

III. OPTIMIZING THE NEGATIVE SELECTION ALGORITHM

This section gives a general description of the optimization scheme along with implementation details.

A. Optimizing the Training Algorithm

In the pseudo code found in Figure 1, the function matches() is used to compare a new detector with every sample in the data set. This function is used heavily to find detectors that do not match any samples in the subset of samples in a data set not in the "self" class. This is because the detector is applied one-by-one to each sample in the training set.

Instead of comparing a new detector with the set of self samples individually, in the optimized training algorithm the comparison proceeds feature-by-feature. This process is best explained as a "filtering" process that is applied on each feature in the data set individually. The filtering process discards the self samples that do not match the new detector in the current feature being processed. The set of self samples becomes smaller and smaller as the filtering proceeds, speeding up the comparison as the features are processed. If there are any self samples remaining in the set after all of the features are processed, we know that the proposed detector matches one or more self samples, and is therefore discarded (as with normal Negative Selection). If there are no self samples left in the set at any point in the processing, we know that the proposed detector does not match any self samples, and can therefore be added to the set of detectors.

B. Optimizing the Classification Algorithm

As in Figure 1, in the pseudo code found in Figure 2 the function matches() is used to compare the set of detectors with the sample to be classified. This function is used to find detectors that match the sample to be classified. It works by comparing each detector in the set of detectors to the sample to be classified individually.

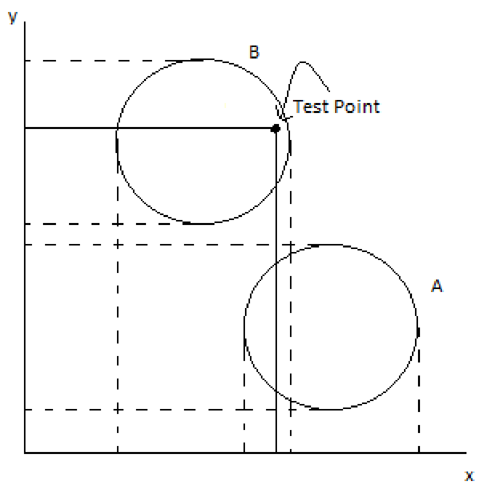
In the same way as the training algorithm, when classifying a sample into self or non-self, the comparison between the sample and the set of detectors proceeds feature-by-feature. The set of detectors that could match the self-sample becomes smaller and smaller as more and more detectors are "filtered" from the set. If the set of detectors is emptied during this filtering process, then the sample does not belong to the self class. If there are detectors remaining in the set after all of the features are processed, then we know that the sample belongs to the non-self class.

Figure 3 shows primary filtering happening in two dimensions with two hyper-sphere detectors. It can be seen that the point to be classified falls within the radius of hyper-sphere B. Filtering based on the feature X, both hyper-spheres would be kept in the set, since the example falls within both in that dimension. Filtering based on the Y dimension would filter out the A detector, since the example does not fall within the radius of the A detector in that dimension.

C. Implementing the Algorithm

The Negative Selection algorithm implemented to test the optimization proposed in this research is implemented using hyper-sphere detectors. The detector radius is defined using Euclidian distance. Each detector contains a center point, defined as a set of coordinates in Euclidian space, as well as a radius. A detector matches a sample only if the sample falls within the

Figure 3. Primary Filtering



radius of the detector. Each detector has a fixed radius, given to the training algorithm as a parameter. The only other parameter needed by the training algorithm is the size of the detector set to be generated. The range of the values in each feature of the data set was normalized to the range [0,1]. This is done to simplify the code, but is not necessary and the optimization can be implemented without this step.

The training algorithm uses a two-stage filtering process to speed up the comparison between the new detector and the set of samples containing the self class. The first stage compares each feature of the sample to the allowed range of the detector in that feature. The allowed range is calculated by adding and subtracting the radius from the coordinate of the center point in that dimension. If the sample does not fall within the range calculated, it is removed from the set. In this manner, the set of samples that could be contained by a detector is iteratively reduced in size.

The secondary filtering step is necessary in this case because of the "roundness" of the hyper-sphere detectors. The primary filtering process could leave some samples in the set if they fall within the hyper-cube that contains the hyper-sphere that is the detector. Secondary filtering then proceeds as normally done by the Negative Selection algorithm, by iteratively comparing the remaining sample set with the detector. After primary and secondary filtering are completed, the samples remaining in the set are the ones that fall within the detector radius. If the set is empty, then the detector can be added to the set of detectors, since it does not match any samples in the self set. If at any point in the primary filtering process the set of samples becomes empty, then the algorithm is able to add the detector immediately, since it is known that the detector does not match any sample in the self set without having to perform secondary filtering. The pseudo code for the optimized Negative Selection training algorithm can be found in Figure 4. In this pseudo code listing the primary filter section contains while loop that is filtering out all detectors that do not meet the criteria. This filter applies the logic described in the previous sections.

In the pseudo code, p["center"] defines the center point, p["radius"] defines the radius, and p["class"] defines the class that the hypersphere belongs to, all attributes of hypersphere p. In the same way, i["data"] defines the center point, and i["class"] defines the class that the point belongs to, all attributes of point i.

Figure 4. Pseudo Code for the Optimized Negative Selection Training Algorithm

```
Definitions:
training_set: a list of the training data points, each with an attached class label
detectors: the set of detectors to be created
population_size: the size of the desired population of detectors
self_class_label: the label of the class designated as "self"
normalize(): a function to normalize the data set
generate_random_antibody(): a function to generate a random antibody
distance(): a function for calculating the Euclidian distance between points
nd: total number of dimensions in the data set

Initialization:
training set = normalize(training_set)
detectors = {}
Training Algorithm:
WHILE | detectors | < population_size:
    self_class = { s | s ∈ training_set AND s["class"] = self_class_label }
    na = generate_random_antibody()

    #primary filtering
    d = 0
    WHILE d < nd
        self_class = { s | s ∈ self_class
        AND na["center"][d] > (s["data"][d] - na["radius"] )
        AND s["data"][d] < (na["center"][d] + na["radius"] )}
        d = d + 1
    ENDWHILE

    #early decision
    IF | self_class | = 0
        detectors = detectors ∪ na
    #secondary filtering
    ELSE
        flagged = FALSE
        FOREACH { s | s ∈ self_class }
            IF distance(na["center"], s["data"]) < na["radius"]
                flagged = TRUE
            ENDIF
        ENDFOREACH
        IF flagged = FALSE:
            detectors = detectors ∪ na
        ENDIF
    ENDIF
ENDWHILE
```

When using a detector in the pseudo code, d["center"] references the vector that contains the center point of the detector d, and d["center"][0] references the first dimension of that vector. Similarly, d["radius"] references the scalar value that defines the radius of the detector. When using a data point in the training set in the pseudo code, s["class"] references the category that the sample s belongs to. Also, the vector that defines the sample s is stored in s["data"], with s["data"][0] referencing the first dimension of the data vector of s.

The filtering process is very similar in the classification algorithm as it is in the training algorithm, only it is done in reverse. Instead of comparing the set of self samples with one detector, the filtering process compares a set of detectors with one sample. The set of detectors is iteratively reduced, and the remaining detectors are subjected to secondary filtering. If a detector remains in the set after both primary and secondary filtering are complete, then the sample is classified as non-self, since it is "matched" by one or more detectors, otherwise it is classified as self. The pseudo code for the optimized Negative Selection classification algorithm can be found in Figure 5. In this pseudo code listing the primary filter section contains while loop that is filtering out all detectors that do not meet the criteria. This filter applies the logic described in the previous sections.

In both of the optimized training and classification algorithms an "early decision" can be made. This happens when the set of self samples is emptied, in the training algorithm, or when the set of detectors is emptied, in the classification algorithm.

IV. TESTING THE OPTIMIZED ALGORITHM

This section contains details about the way in which the optimization was implemented, along with the data set used and the claims being tested.

To test the optimization, the Breast Cancer Wisconsin (Diagnostic) Dataset was chosen from the UCI repository [19]. This data set was chosen because it is limited to two classes, which fits with the AIS paradigm. To test the algorithm, some preprocessing was done to the data set. The class label of each sample was placed in the first column of the data set. All samples with missing values were removed from the data set, making the data set smaller but simplifying the algorithm. All duplicate rows were removed from the data set as well. After this was done, the data set contained 683 labeled samples, each with 9 real-valued features. The class labels found in this data set are "malignant" with 239 samples found in the data set, and "benign" with 444 samples found in the data set.

The model was validated using 10-fold cross validation. To do this, the dataset used was split evenly into 10 subsets. From these 10 subsets, training, validation, and testing sets are created. The training set created used 80% of the samples, the validation 10%, and the testing set 10% of the data. Stratification was also used, which is a technique used to make sure that each of the 10 subsets is created so that it contains the same proportion of each class in the data set. Through this process, we are able to create 10 unique testing sets, 10 unique validation sets, and 10 unique training sets. By cycling through these, the experiments are performed 10 times and the results are averaged.

Figure 5. Pseudo Code for the Optimized Negative Selection Classification Algorithm

```
detectors: set of the detectors generated by the training algorithm
x: the sample to be classified
self_class_label: label of the class designated as "self"
non_self_class_label: label of the class designated as "non-self"
distance(): a function for calculating the Euclidian distance
between points
nd: total number of dimensions in the data set

Classification Algorithm:
#primary filtering
d = 0
WHILE d < nd
    detectors = { a | a ∈ detectors
    AND x["data"] [d] > (a["point"][d] - a["radius"])
    AND x["data"] [d] < (a["point"][d] + a["radius"]) }
    d = d + 1
ENDWHILE

#secondary filtering
FOREACH {a | a ∈ detectors}
    d = distance(x["center"], a["center"])
    IF d <= a["radius"]
        return non_self_class_label
    ENDIF
ENDFOREACH
return self_class_label
```

Since the algorithm requires one parameter, we set aside one subset in every test run to determine the best values for these parameters. To accomplish this, a grid search is performed on the validation set, with the objective of finding the value for the parameter which maximizes the accuracy of the algorithm. The parameter is the radius of the hyper-spheres. The radius is varied from 0.01 to 0.99 in 0.01 increments

The tests were performed on an Intel i5 processor running at 1.80 GHz, with 4 GB of memory. The operating system used was 64-bit Windows 8.1.

Four experiments were performed with the original Negative Selection algorithm and the optimized version of the Negative Selection algorithm. The results of the experiments are detailed in the next section. The experiments are designed to demonstrate three claims that are made about the optimized Negative Selection algorithm. The claims deal with the execution time, classification time, and classification performance of the algorithm. Our claims about the algorithm are these:

1. The optimized training algorithm is faster than the unoptimized training algorithm.
2. The optimized classification algorithm is faster than the unoptimized classification algorithm.
3. The optimization does not affect the accuracy of the algorithm, being functionally the same.

To make the comparisons between the optimized and unoptimized algorithms as unbiased as possible, two methods were used: when testing the training algorithm, both versions of the algorithm were given the same parameters and the exact same data set, with the same set of sub data sets (due to the 10-fold cross validation). When testing the classification algorithm, the exact same set of detectors is provided to both versions of the algorithm. This was done so that we could compare both versions of the algorithm without worrying about randomness affecting the results.

When comparing the accuracy of the optimized and unoptimized algorithms, the accuracy is calculated as follows:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)) \qquad (1)$$

were TP is the number of true positive predictions, TN is the number of true negative predictions, FP is the number of false positive predictions, and FN is the number of false negative predictions.

All experiments were performed on an Intel i5-based computer running at 1.80 GHz. The computer has 4 GB of memory, and the operating system used is 64-bit Windows 8.1. Both the optimized and unoptimized algorithms were coded in Python 3.4.

V. EXPERIMENTAL RESULTS

This section shows the results of the experiments and demonstrates the validity of the claims made in the previous section. To simplify the tests, the detector radius was set to 0.5 for the experiments graphed in Figures 6, 7, and 8. This radius was found using a grid search, which was used to find the detector radius that maximized the accuracy of the algorithm. The grid search was performed using the validation set.

The relationship between the training time and the data set size is shown in Figure 6. The detector set size is held constant at 1000, and the data set size was increased from 100 to 500. It can be seen that the optimized training algorithm remains linear on the number of samples in the data set. The time is measured in seconds.

A confidence interval was calculated using data from Figure 6 for the difference in the average time taken to finish by both algorithms. To do this, 10 data points were taken from the last test graphed in the figure. For this test, the detector set size was 1000, and the data set size was 500. With these values a confidence interval was calculated at the 95% confidence level. The difference in the average time taken to complete training was calculated to be between 1.3 and 1.01 seconds, with the optimized algorithm being faster. This helps to show that claim 1 is true.

The relationship between the training time and the detector set size is shown in Figure 7. The data set size is held constant at 500, and the detector set size was increased from 100 to 1000. The optimized training algorithm is also linear with the number of detectors in the set. It is faster than the unoptimized training algorithm.

A confidence interval was calculated using data from Figure 7 for the difference in the average time taken to finish by both algorithms. To do this, 10 data points were taken from the last test graphed in the figure. For this test, the detector set size was 1000, and the data set size was 500. With these values a confidence interval was calculated at the 95% confidence level. The difference in the average time taken to complete was calculated to be between 1.07 and 1.04 seconds, with the optimized algorithm being faster. This also helps to demonstrate the validity of claim 1.

The relationship between the size of the set of detectors and the classification time is shown in Figure 8. The classification time is the time taken to classify one sample. The size of the detector set was increased from 100 to 1000.

The confidence interval was calculated using data from Figure 8 for the difference in the average time taken to finish by both algorithms. To do this, 10 data points were taken from the last test graphed in the figure. For this test, the detector set size was 1000. With these values a confidence interval was calculated at the 95% confidence level. The difference in the average time taken to complete was calculated to be between 0.0033 and 0.00033, with the optimized algorithm being faster. The results of this experiment demonstrate that claim 2 is valid.

The fourth experiment is done on the training and classification algorithms in tandem, proving that the combination of the optimized training and classification algorithms does not negatively affect the accuracy of the algorithm. Figure 9 shows the relationship between the data set size and the accuracy achieved by the algorithm. As mentioned, the optimized and unoptimized versions of the algorithm use the exact same data set to create the set of detectors. The size of the set of detectors generated is held constant at 1000. Although it is not easily seen, the accuracy achieved by the optimized algorithm does not match the accuracy of the unoptimized algorithm exactly. This is due to the fact that the Negative Selection algorithm uses randomness in the training process.

A t-test was performed to compare average accuracy achieved by both the optimized and unoptimized algorithms. The samples were paired according to the data set size used, using the same data that is graphed in Figure 9. The confidence level used was 95%. The null hypothesis could not be rejected, meaning that the analysis did not provide evidence against claim 3. Additionally, the Pearson correlation between the paired accuracies was calculated to be 0.996, a value that shows that the accuracies of the unoptimized and optimized versions of the algorithm are very closely related.

VI. CONCLUSIONS AND FUTURE WORK

The optimized training algorithm is, on average, 6.6 times faster than the unoptimized training algorithm, when averaging the results from Figures 7, and 8. The optimized classification

Figure 6. Data Set Size and Training Time

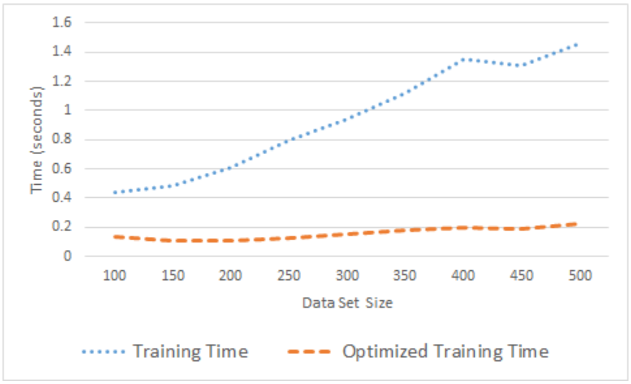


Figure 7. Training Set Size and Training Time

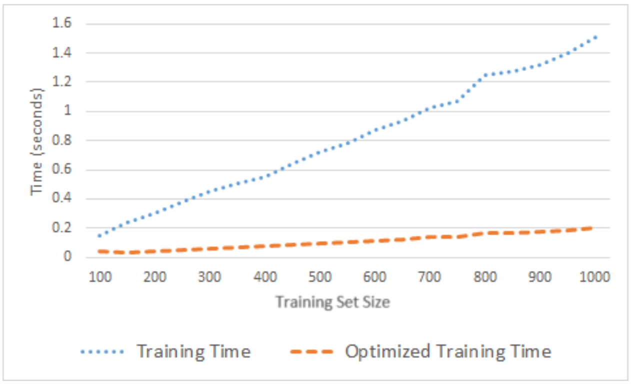


Figure 8. Antibody Set Size and Classification Time

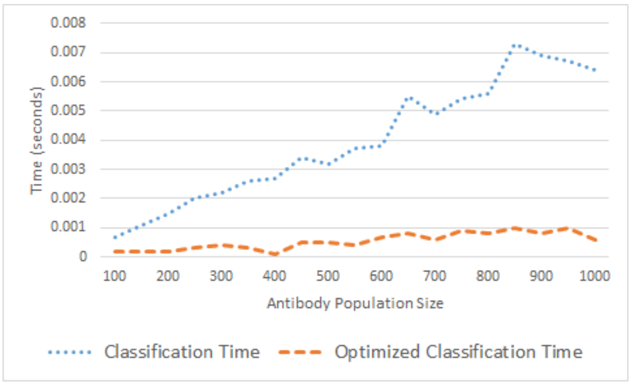
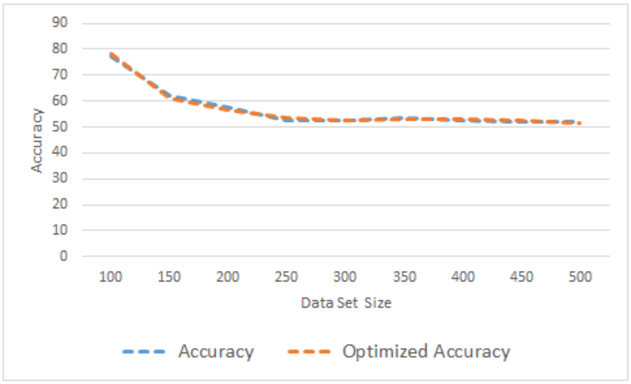


Figure 9. Data Set Size and Accuracy



algorithm is, on average, 11.3 times faster than the unoptimized classification algorithm, when averaging the results from Figure 9. Furthermore, the experiments performed have shown that the accuracy of the algorithm does not diminish when the optimization is applied.

The optimized algorithm, however, has several weaknesses. First, the complexity of the algorithm remains in the same class, although the constants are decreased significantly. Second, the optimization is only applicable to detectors in which each dimension can be evaluated individually, and which allow the set of data points or detectors to be filtered. That is, the optimization works on detectors types that allow a data point or detector to be taken out of the set if it does not match in one individual dimension, this is not always possible. Third, the density of the points in the data set can have a significant effect on the performance of the optimized training algorithm. If the points are densely packed, then none of them will be filtered out by the primary filtering process and the secondary filtering will then have to perform all of the work. In future research, this optimization scheme could be applied to other data sets to highlight the effect that the density of the data set has on the performance of the optimized algorithm.

All previous research we have found on the optimization of the Negative Selection algorithm has applied to detectors other than hyper-spheres. This research has demonstrated a simple way to optimize the performance of the Negative Selection algorithm when hyper-spheres are used. A direct comparison between this optimization scheme and others found in the literature is not possible.

Although the optimization has been demonstrated experimentally to not affect the accuracy of the algorithm, this has not been proven formally. Future research could be completed to provide a formal proof of the optimized Negative Selection algorithm's equivalency with the unoptimized Negative Selection algorithm. Future work can also be done in the application of the optimization to more complex data sets and exploring the performance of the optimized algorithm. Lastly, the optimization proposed in this publication could be very useful when combined with negative databases, described in [20] by Esponda et al.

REFERENCES

[1] Kim, J. and Bentley, P. Negative selection and niching by an artificial immune system for network intrusion detection. in *Proceedings of GECCO'99,* (Orlando, Florida, USA, 1999), 149-158.

[2] Forrest, S., Perelson, A. S., Allen, L., and Cherukuri, R.. Self-nonself discrimination in a computer. in *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Washington, DC, USA, 202.

[3] Dasgupta, D., & Nino, F. *Immunological computation: theory and applications*. CRC Press, 2008

[4] Dasgupta, D., & Majumdar, N. S. Anomaly detection in multidimensional data using negative selection algorithm. in *Proceedings of the World on Congress on Computational Intelligence,* 2002, 1039-1044.

[5] Balthrop, J., Esponda, F., Forrest, S., and Glickman, M. Coverage and Generalization in an Artificial Immune System. in *GECCO*, (New York, 2002), 3-10.

[6] Ayara, M., Timmis, J., De Lemos, R., and Forrest, S. Immunising automated teller machines. in *Artificial Immune Systems,* Springer Berlin Heidelberg, 2005, 404-417

[7] Percus, J., Percus O., and Perelson A. Predicting the Size of the T-Cell Receptor and Antibody Combining Region from Consideration of Efficient Self-Nonself Discrimination. in Proceedings of National Academy Of Sciences USA, 1993, 1691-1695

[8] Cserey, G., Porod, W., and Roska, T. An artificial immune system based visual analysis model and its real-time terrain surveillance application. in *Artificial Immune Systems*, Springer Berlin Heidelberg, 2004, 250-262.

[9] Şahan, S., Polat, K., Kodaz, H., and Güneş, S. The medical applications of attribute weighted artificial immune system (AWAIS): diagnosis of heart and diabetes diseases. in *Artificial Immune Systems*, Springer Berlin Heidelberg, 2005, 456-468.

[10] Elberfeld, M., and Textor, J. Efficient algorithms for string-based negative selection. in *Artificial Immune Systems,* Springer Berlin Heidelberg, 2009, 109-121

[11] Liśkiewicz, M., and Textor, J. Negative selection algorithms without generating detectors. in *Proceedings of the 12th annual conference on Genetic and evolutionary computation* ACM, (Portland, OR, 2010), 1047-1054.

[12] Elberfeld, M., and Textor, J. Negative selection algorithms on strings with efficient training and linear-time classification. *Theoretical Computer Science*, *412*(6), 534-542.

[13] Textor, J. Efficient negative selection algorithms by sampling and approximate counting. in *Parallel Problem Solving from Nature-PPSN XII*, Springer Berlin Heidelberg, 2012, 32-41.

[14] Wang, D., Xue, Y., and Yingfei, D. Anomaly Detection Using Neighborhood Negative Selection. *Intelligent Automation & Soft Computing*, *17*(5), 595-605.

[15] Wang, D., Xue, Y., and Dong, Y. NNS: A Novel Neighborhood Negative Selection algorithm. in *World Automation Congress (WAC), 2012*, IEEE, 2012, 453-457.

[16] Yang, T., Deng, H. L., Chen, W., and Wang, Z. GF-NSA: A Negative Selection Algorithm Based on Self Grid File. *Applied Mechanics and Materials*, *44*, 3200-3203.

[17] Wen, C., Xiaoming, D., Tao, L., and Tao, Y. Negative selection algorithm based on grid file of the feature space. *Knowledge-Based Systems*, *56*, 26-35.

[18] Ji, Z., and Dasgupta, D. V-detector: An efficient negative selection algorithm with "probably adequate" detector coverage. *Information sciences,* vol. 179, no. 10, 2009, 1390-1406.

[19] Bache, K. and Lichman M. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml], Irvine, CA: University of California, School of Information and Computer Science. Accessed December 3, 2014.

[20] Esponda, F., Ackley E., Forrest S., and Helman P. Online negative databases. in *Artificial Immune Systems*, Springer Berlin Heidelberg, 2004, 175-188.