World Journal of Science and Technology Research
Vol. 5, No. 1, December 2017, pp. 1-14, ISSN: 2329 - 3837 (Online)
Available online at http://wjst.org/

Research article

Membership Function Sensitivity Using a Fuzzy Genetic Algorithm for Intrusion Detection

Fred Guyton

College of Engineering and Computing

Nova Southeastern University

Fort Lauderdale, FL

1-678-575-9055

E-mail: fg292@mynsu.nova.edu

Abstract

Intrusion Detection Systems (IDS) are a key component of system security, used to identify malicious activity on networks and host machines, and much research is on-going to determine the best techniques for optimization. A program was developed to study intrusion detection consisting of a genetic algorithm with fuzzy logic used for encoding which incorporated a triangular membership function. A sensitivity study was conducted to understand how the variation in the granularity of the number of overlapping membership functions affected the overall effectiveness of the system.

The results indicated that on the course end of granularity there is little effect, but at some point (in this case after five overlapping member functions) the effectiveness started to trail off at a rate of seven percent per additional member function. Proposed future work arising from this study is also discussed. **Copyright © WJSTR, all rights reserved.**

Keywords: Intrusion Detection System (IDS), Genetic Algorithm (GA), Fuzzy Logic, Membership Function, Anomaly Detection

Introduction

The world is increasingly networked. No longer is it just computers, but today with wireless networks our smartphones talk to our tablets, to our security systems to our appliances. And every new component type joining the fray brings with it additional security threats. One device in the arsenal defending against those threats is the Intrusion Detection System (IDS).

From a top level, IDSs are typically defined by where they monitor data to look for intrusive behavior, namely network-based or host-based. Network-based systems monitor network traffic in devices such as routers, network interface cards, etc., whereas host-based IDSs monitor file and process activity from the applications on a specific system.

A second level of IDS categorization is misuse and anomaly, referring to the type of cyber attack the system is meant to detect. Systems built for misuse detection monitor for known types of attacks with a signature database containing information on previously observed attacks, very similar to virus detection software. The advantages of misuse-based IDSs are they are fast and efficient (in terms of accuracy of detection). The major shortcoming is that by design they will not detect new types of attacks because there is no signature of that attack in the database.

Systems built for anomaly detection look for system behavior deviating from what is deemed normal. This is done by modeling the activity of the environment and comparing observed behaviors to what the model defines as "normal." The advantage of anomaly-based systems is that they detect new types of attacks, but the downside is that there can be excessive alerts for non-threating behavior due to that behavior being new and outside of the model guidelines for normal. Additionally, the speed of operation for these systems can be problematic when they have to constantly evaluate very large amounts of data.

There is a third approach typically referred to as a hybrid model where anomaly-based systems are used to develop signatures that are then used in misuse-based IDSs. The concept is to get the best of both types of systems by using a signature approach for speed and a model approach for zero-day awareness.

Buczak and Guven [1] conducted an extensive literature survey with regard to the on-going work to improve anomaly-based detection using data mining and machine learning techniques. They identified and categorized IDS research into twelve areas: association rules and fuzzy association rules, decision trees, ensemble learning, clustering, naïve Bayes, neural networks, inductive learning, Bayesian networks, sequential pattern mining, evolutionary computing, support vector machines and hidden Markov models. For evolutionary computing, of interest herein, [1] identified six branches of work: evolution strategies, particle swarm optimization, ant colony optimization, artificial immune systems, genetic programming and genetic algorithms.

Genetic algorithms (GA) are inspired by evolution and survival of the fittest. As defined by two of the pioneers in genetic algorithms, John Holland and David Goldberg, GA's are "probabilistic search procedures designed to work on large spaces involving states that can be represented by strings" [2].

GA's model a problem optimization solution as a chromosome and each data element of the chromosome is referred to as a gene. The effectiveness of a chromosome (solution) is defined as fitness. Through many generations chromosomes evolve; those that are a better fit survive to reproduce while others die out.

Figure 1 is a high-level view of a genetic algorithm.The process starts with an initial population of chromosomes. This could be any number but a typical size is 200. Each of these chromosomes is supposed to represent a potential solution to the problem under study, but since this is the first iteration this group is usually created using random numbers.

The next step is that each chromosome is evaluated by the defined fitness formula and given a fitness score. If the chromosome with the highest score meets some predefined optimization rule then the program is done. If not then this population of chromosomes is ranked by fitness score and some subgroup of high performers is selected to carry on the species. Pairs in the subgroup are then allowed to reproduce. This consists of using part of each parent to create a new chromosome (child) just as in real life each parent contributes to the genes of their child. And also like in real life, occasionally through random chance a gene in a child mutates resulting in a gene that came from neither parent but providing an opportunity for divergence that may result in higher fitness for survival. The subgroup continues having children until a completely new population has been formed.

This next generation is then evaluated for fitness and the process continues for generation after generation until the optimization threshold is met or until a predefined maximum number of generations is reached.

Materials and Method

Background

Applying genetic algorithms to anomaly-based intrusion detection was performed as early as 1995 by Crosbie and Spafford [3]. They used multiple agent technology with each agent monitoring one network parameter and a GA used to find a set of agents that together identified network anomalous behavior. The agent concept was interesting but ultimately suffered from time consuming training and communication issues.

In 2000, Bridges and Vaughn [4] used genetic algorithms to tune their fuzzy membership functions for anomaly-based intrusion detection using fuzzy data mining techniques, but the GA was not used to actually converge on an optimized IDS rule.

Then in 2001, Chittur [5] took a more generic approach and developed a genetic algorithm for anomaly detection which he tested on DARPA's KDD Cup dataset [6]. This work demonstrated that a GA was able to generate a model with high detection rate capability and acceptable false positive rate for intrusion detection.

As the work of applying genetic algorithms to network audit data grew, the question of what data to monitor grew equally. In 2005 Chebrolu, Abraham and Thomas [7] introduced the idea of feature selection to intrusion detection. Their assertion was that all data parameters available to intrusion detection analysis were not equal in terms of value to the solution and that reducing the data under consideration would improve the systems' effectiveness.

Toosi and Kahani [8] introduced fuzzy logic into the combination of genetic algorithms and intrusion detection in 2006. They used a five-input, single-output of Mamdani fuzzy inference system with two Gaussian member functions for input classification. Training and testing using the dataset from [6] their model demonstrated moderate to high accuracy depending on the type of attack.

In 2009, Yunwu [9] combined a fuzzy expert system and GA approach for intrusion detection. The expert system used a Fuzzy C-Means (FCM) algorithm to fuzzify input values and Mamdani's minimum fuzzy implication rule for approximate reasoning. Also using the dataset from [6] experimental results were presented showing reasonable effectiveness.

Jongsuebsuk, Wattanapongsakorn and Charnsripinyo [10] developed a system in 2013 using a fuzzy genetic algorithm with a trapezoidal membership function, in addition to feature selection to minimize compute time. They validated their model using training data from their university network and testing it at the same in real-time resulting in detection rates over 90% for various types of attacks.

In 2014 Pal and Parashar [11] used a fuzzy genetic algorithm approach in a model combining two techniques to improve the GA accuracy: feature selection based on work published by Abraham, Jain, Thomas, and Han [12], and the fuzzy logic with a triangular membership function as described by Yunwu [9]. Their work investigated the sensitivity of feature selection showing that for their algorithm, optimum selection converged at 15 specific parameters out of the 41 in the KDD dataset [6].

Problem Statement

Significant research is being performed in the academic community using genetic algorithms in anomaly-based intrusion detection. As presented in the prior section, many researchers use fuzzy logic to improve the model detection rate. Additionally, quite a number of studies use feature selection to reduce the amount of data being consumed and processed by the model with the goal of improving speed. Both techniques were used by [11] and they went so far as to study the sensitivity of various levels of feature selection. However, there was no like analysis of the efficacy of changes to the membership function used in the fuzzy logic. Model performance could be improved if a more effective membership function were available, so the sensitivity of such model changes should be investigated.

Objective of Study

The objective of this work is to investigate how changes to the membership function affected the accuracy of the model used by Pal and Parashar [11]. The reference work used a model with five overlapping triangular membership functions. The current research is to determine how variations in the number of overlapping membership functions affects accuracy.

Evolve Framework

Evolve is the name of the program developed as part of this research. It is written in Java using the Eclipse IDE and executed in a Microsoft Windows 10 environment, although given the cross-platform nature of Java, Evolve would run as well on Linux. The system consists of three modules:

- evolvePrep – Encoding program.
- evolve – Learning program.
- evolveAssess – Testing program.

Encoding

Encoding the data is a multipass process which starts by converting symbolic data to digital data. For example, a parameter such as protocol_type with valid settings of icmp, udp and tcp, would be converted to 0, 1 and 2 respectively.

The second step is data normalization. In this process, the continuous (non-symbolic) data is normalized so that all parameters have a maximum value of 10.

The third and final step in encoding is fuzzification. The program is designed to use a triangular membership function specified by:

$$f(x;a,b,d) = max\left(min\left(\frac{x-a}{b-a},\frac{d-x}{d-b}\right),0\right)$$

where the triangular curve is a function of vector x, and has three scalar parameters {a, b, c} that determine the coordinates of the three corners of the triangle. The parameters a and c locate the two base angles of the triangle and the parameter b locates the peak.

Aside from input/output filenames, evolvePrep has one control variable that is part of the input configuration data which is the number of triangular membership functions being used, a key component of the study.

Learning

Dataset

To test the ability of this model and associated variables, the KDD'99 Cup dataset [6] was chosen. This dataset consists of 41 features of which 34 are continuous and seven are symbolic or boolean. The size is approximately five million records with 20% of those representing normal operations and the remaining 80% representing one of four types of network attacks: denial of service (DOS), remote to local (R2L), network probing (Probe) and user to root (U2R).

For the training phase, a ten percent random sample of the KDD'99 Cup dataset was used.

Feature Selection

Not all 41 features of the database provide the same level of information gain and so the Evolve framework is designed to allow easy modification to the features selected for a given study. Simply flipping the bits in a binary column of the geneDescriptor input file causes the feature to be selected (if =1) or ignored (if=0).

In terms of how this selection maps into the genetic algorithm, each feature selected represents a gene and all the genes together represent a chromosome as discussed in the following section.

Genes and Chromosomes

To account for the variability of feature selection and keep the gene and chromosome formats consistent, each gene is represented by two characters and specified in base 10 number system thus allowing a range of 0 – 99. The chromosome is created by concatenating all of the genes as shown in Figure 2 which is a selected representation of an Evolve chromosome with 14 genes.

Genetic Algorithm

The flowchart for the genetic algorithm used by Evolve is presented in Appendix A. The program has three main inputs: four control variables via a configuration file, the gene descriptor which handles feature selection, and the encoded data for training.

The four control variables which can be modified for the desired settings are: 1) population size (sizePopulation), 2) the top performing percentage of the population that is selected for reproducing and thus creating the next generation (pctReproduce), 3) the mutation rate used in the "birth" process (mutationRate) and 4) the maximum number of generations for which the program will run (maxGenerations).

As depicted in the flowchart, after initial input the program has the option to restart by reading in a restart file, which consists of the last population that was in use at the time of shutdown. Otherwise an initial population is created. Based on the input parameter sizePopulation (n), the process will create *n* chromosomes similar to those shown in Figure 2 with all the genes filled with randomly generated numbers. This is the first generation.

At this point a loop is entered that will continue until the maximum number of generations have been met. Inside this loop is the process that allows the evolution of the chromosomes to an optimal solution.

The first step in the evolutionary process is evaluating the fitness of each member of the population. For each chromosome, a fitness value is calculated. Fitness is calculated by the following formula:

$$fitness = \left[\frac{\alpha}{A}\right] - \left[\frac{\beta}{B}\right]$$

where: α is the number of correctly detected attacks, A is the total number of attacks in the training dataset, β is the number of false positives (normal connections falsely identified as attacks) and B is the total number of normal connections in the dataset.

After determining the fitness value for each member of the current population, the chromosome with the highest fitness score becomes the current candidate for the optimum solution. A check for convergence is made by comparing the chromosome to the prior iteration's top scorer and if it is the same then the convergence counter is incremented. When the convergence counter exceeds 20, i.e., the solution has not changed in 20 generations, the solution is deemed converged and exits.

If convergence has not been achieved, then the next step is the selection of the population that will be allowed to reproduce. These chromosomes will be used to create a completely new, or evolved, population that theoretically should be better that the parent population.

The input parameter pctReproduce (percent to reproduce) specifies the size of the reproducing population relative to the entire population. For example, if the population size is 100 and the pctReproduce = 0.35 then the top 35% performers for the population, or top 35 chromosomes, will be allowed to reproduce. The bottom 65% of the chromosomes in the population will be discarded.

In another loop that continues until the new population size is the same as the prior population size, two chromosomes are chosen at random from the selected reproducers. These two "give birth" to a new chromosome and that child is added to the new population.

The process of chromosomes giving birth includes two components: cross-over and mutation. Cross-over is where the two parent chromosomes donate part of themselves to form the child chromosome and the cross-over point is the dividing line. In a case where the cross-over of 50% is used then one half of each parent would be contributed to the child. In Evolve, there is a range of possible cross-over points for each birth. The size of the range is controlled by the input variable pctCrossover. As an example, if the cross-over value of 0.40 or 40% is used, the cross-over range is determined by:

$$CO = (\ cPct,\ 1 - cPct\ )$$

$$CO = (\ 0.30,\ 1 - 0.30\ ) = (\ 0.30,\ 0.70\ )$$

where CO is the cross-over point and cPct is the input value pctCrossover. For the case of 30%, the range goes from 30% to 70%.

For each birth, a random number is selected from within that CO range for the actual cross-over point used. Figure 3 is an example of a 60% cross-over birth.Notice that in the top chromosome the first eight genes are contributed and in the second chromosome the last six genes are contributed.

The second step in birth of a new chromosome is mutation. For each birth a random number from 0 to 1 is generated. If that number is less than the mutation rate (specified by the input value of mutationRate) then the chromosome is selected for mutation. At that point a random gene is selected and a random number within that gene's range will be inserted.Figure 4 depicts the third gene of the chromosome mutating from 09 to 07 as part of the birth process.Once enough births have occurred to create a new population the cycle begins again at the fitness check continuing until the maximum number of generations has been reached or convergence achieved.

Testing

The third section of Evolve is the testing program evolveAssess. This component inputs the geneDescriptor described earlier to understand the encoded data as well as the structure of the chromosome. Then the system inputs a single chromosome, presumably one that represents an optimum solution as determined by the genetic algorithm.

This chromosome is then used against the test dataset to determine its ability to predict an attack. Each record in the dataset represents either normal operation or an attack. The chromosome is an attack indicator, so for each record in the dataset, that record and the chromosome are compared.

Figure 5 is the flowchart for the scoring. The comparison is deemed either a chromosome match or not. If there is a match, then the program is predicting an attack. If the data record indicates there is an attack, then it is a correct prediction. If the data record indicates there is no attack, then the prediction represents a false positive.

Conversely, if there is no match between the chromosome and the data record, the program is predicting normal operation. If the data record indicates there is an attack, the prediction represents a false negative. If the data record indicates there is no attack, then it is a correct prediction of normal operations.These individual scores are accumulated for the entire dataset resulting in an overall score for the chromosome being tested.

Results and Discussion

The Evolve framework was used to study the sensitivity of varying the number of overlapping membership functions used in the fuzzification of the data.

Feature Selection

For this work, the features selected by Pal and Parashar [11] were used and are presented in Table 1.

Fuzzification

The purpose of the study is to determine the effect of varying granularity in the membership function. The baseline is a five member set also as selected by [11] and shown in Figure 6.

All variability data is normalized against the baseline so as to focus on the sensitivity function.

Genetic Algorithm Settings

The settings used to control the genetic algorithm for the current work are presented in Table 2.

In general, one would expect these settings to affect the speed of convergence or even lack of convergence as opposed to the actual attained optimum solution. In the early stages of the development of Evolve several sensitivity studies were conducted around these three inputs (not reported herein) which confirmed the hypothesis on convergence effect.

However another issue that must be watched for in genetic algorithms is premature convergence to local optima. The problem usually occurs when a lack of genetic diversity develops in a population. The approach taken by Evolve in cross-over and mutation is meant to combat this problem but multiple successive runs of identical input did result in an occasional premature convergence. To neutralize the problem, all solutions were run at least three times to ensure convergence to a maximum.

Sensitivity Runs

Evolve was run at four variations of the overlapping membership function as compared to the baseline.

Baseline = 5

Variations = (2, 8, 11, 14)

The plot of 2 and 8 are shown in Figure 7. Plots for 11 and 14 are omitted due to the density of the functions on the graph.

Relative to the baseline, the four solutions detection rates were as follow in Table 3.Decreasing the number of member functions from five to two showed no effect. Evolved repeatedly converged to the same solution for both settings. Increasing the number of member functions did have a negative effect. Eight member functions decreased the detection rate by 39%. Interestingly, increasing the number from 8 to 11 showed little change, going from a negative 39% to negative 40%. The solutions were different but the rates were close. Finally, adding an additional set, going from 11 to 14 further decreased the systems effectiveness, this time going down to negative 54%. The results are plotted in Figure 8.

Given the baseline is five, one can look at these results as the effect of decreasing granularity and that of increasing granularity.

Decreasing the granularity resulted in no change in effectiveness. This suggests that there is a point at which granularity is maximized as to detection rate. In this instance one could surmise that five, six or seven overlapping membership functions could be that optimum setting.

Increasing the granularity clearly decreased effectiveness. This is no surprise given the more granular the solution the less generic it becomes and more difficult to find data matches. The interesting aspect of the results are how close eight and eleven are, a decrease of 39% and 40% respectively. One possible answer is that the case of eight member functions did prematurely converge to local optima even though it was run three times.

If that is the case, then the real decrease would be shown by the dashed line in Figure 8 indicating a linear decrease of around 7% per additional member function and pointing to a value of -20% for the case of eight. This is only speculation and needs verification.

Conclusion

A program was developed to study aspects of intrusion detection. The system consists of a genetic algorithm with fuzzy logic used for encoding. The fuzzy logic incorporated multiple overlapping triangular membership functions. A sensitivity study was conducted to determine how the variation in the number of overlapping membership functions affected the overall effectiveness of the system.

The study was baselined at five membership functions. Results showed that decreasing the number down to two provided no change to the effectiveness. Conversely the results did show that increasing the granularity to eight, eleven and fourteen increasingly made the system less effective at roughly 7% per additional member function.

Suggestions for future work include verifying the linear relationship of the decrease in effectiveness. Also, increasing the granularity to determine the real cutoff point at which the decrease begins would be instructive. Two other areas of interest are studying the relationship of the membership function sensitivity to the dataset feature selection, and looking at the sensitivity of using various forms of membership function such as triangular versus trapezoidal.

References

[1] Buczak, Anna L., and Erhan Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection." *IEEE Communications Surveys & Tutorials* 18, no. 2 (2016): 1153-1176.

[2] Goldberg, David E., and John H. Holland. "Genetic algorithms and machine learning." *Machine learning* 3, no. 2 (1988): 95-99.

[3] Crosbie, Mark, and Gene Spafford. "Applying genetic programming to intrusion detection." In *Working Notes for the AAAI Symposium on Genetic Programming*, pp. 1-8. Cambridge, MA: MIT Press, 1995.

[4] Bridges, Susan M., and Rayford B. Vaughn. "Intrusion detection via fuzzy data mining." In *12th Annual Canadian Information Technology Security Symposium*, pp. 109-122. 2000.

[5] Chittur, Adhitya. "Model generation for an intrusion detection system using genetic algorithms." *Internet link: http://www1. cs. columbia. edu/ids/publications/gaids-thesis01. pdf* (2001).

[6] Lippmann, Richard, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. "The 1999 DARPA off-line intrusion detection evaluation." *Computer networks* 34, no. 4 (2000): 579-595.

[7] Chebrolu, Srilatha, Ajith Abraham, and Johnson P. Thomas. "Feature deduction and ensemble design of intrusion detection systems." *Computers & security* 24, no. 4 (2005): 295-307.

[8] Toosi, Adel Nadjaran, and Mohsen Kahani. "A new approach to intrusion detection based on an evolutionary soft computing model using neuro-fuzzy classifiers." *Computer communications* 30, no. 10 (2007): 2201-2212.

[9] Yunwu, Wang. "Using fuzzy expert system based on genetic algorithms for intrusion detection system." In *IFITA'09 International Forum on Information Technology and Applications,* vol. 2, pp. 221-224. IEEE, 2009.

[10] Jongsuebsuk, P., NaruemonWattanapongsakorn, and ChalermpolCharnsripinyo. "Network intrusion detection with Fuzzy Genetic Algorithm for unknown attacks." In *2013 International Conference on Information Networking (ICOIN),* pp. 1-5. IEEE, 2013.

[11] Pal, Dheeraj, and Amrita Parashar. "Improved Genetic Algorithm for Intrusion Detection System." In *2014 International Conference on Computational Intelligence and Communication Networks (CICN),* pp. 835-839. IEEE, 2014.

[12] Abraham, Ajith, Ravi Jain, Johnson Thomas, and Sang Yong Han. "D-SCIDS: Distributed soft computing intrusion detection system." *Journal of Network and Computer Applications* 30, no. 1 (2007): 81-98.

Figure's and Table's

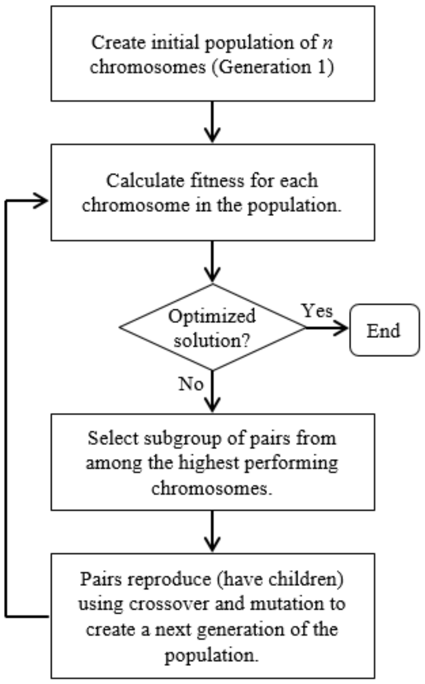
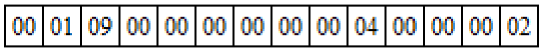
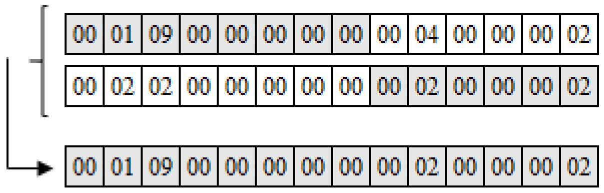

Create initial population of *n* chromosomes (Generation 1)
Calculate fitness for each chromosome in the population.
Optimized solution?
Yes
End
No
Select subgroup of pairs from among the highest performing chromosomes.
Pairs reproduce (have children) using crossover and mutation to create a next generation of the population.

Figure 1: Genetic algorithm flowchart.



00	01	09	00	00	00	00	00	00	04	00	00	00	02

Figure 2: Example chromosome from Evolve.



00	01	09	00	00	00	00	00	00	04	00	00	00	02
00	02	02	00	00	00	00	00	00	02	00	00	00	02
00	01	09	00	00	00	00	00	00	02	00	00	00	02

Figure 3: Cross-over birth at 60%.

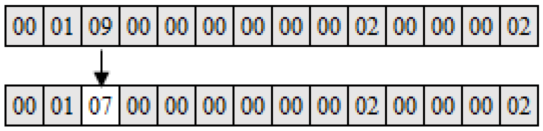


Figure 4: Gene mutation.

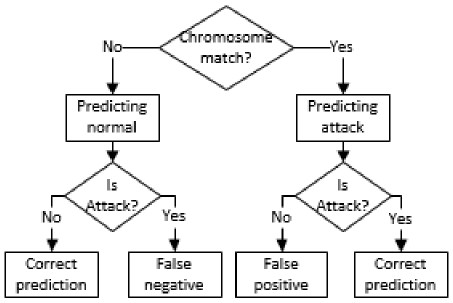


Figure 5: Testing phase scoring algorithm.

Item	Label	Type
1	duration	continuous
2	protocol_type	symbolic
4	flag	symbolic
5	src_bytes	continuous
8	wrong_fragment	continuous
10	hot	continuous
23	count	continuous
25	serror_rate	continuous
26	srv_serror_rate	continuous
34	dst_host_same_srv_rate	continuous
37	dst_host_srv_diff_host_rate	continuous
38	dst_host_serror_rate	continuous
39	dst_host_srv_serror_rate	continuous

Table 1: Dataset feature selection.

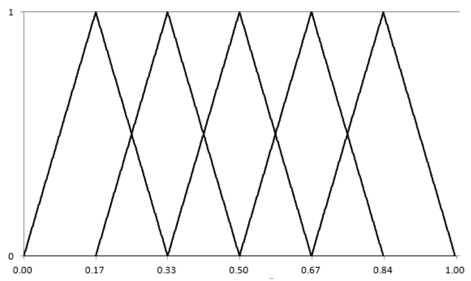


Figure 6: Five Overlapping membership functions.

sizePopulation	200
pctReproduce	0.35
mutationRate	0.05

Table 2: GA Settings

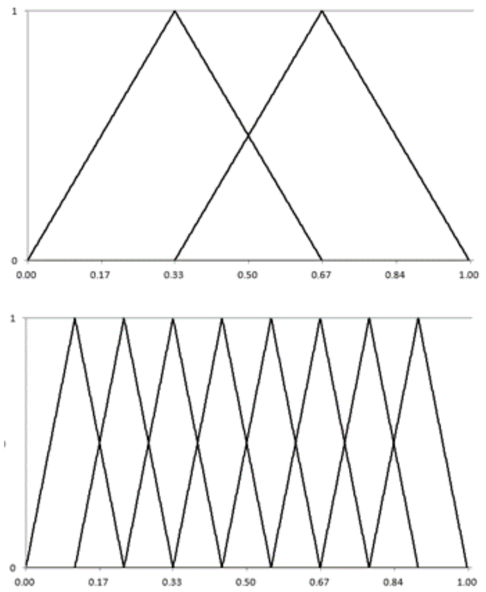


Figure 7: Plots for 2 and 8 membership functions.

n MF	Variation in DR
2	0%
8	-39%
11	-40%
14	-54%

Table 3: Detection rate variation from baseline.

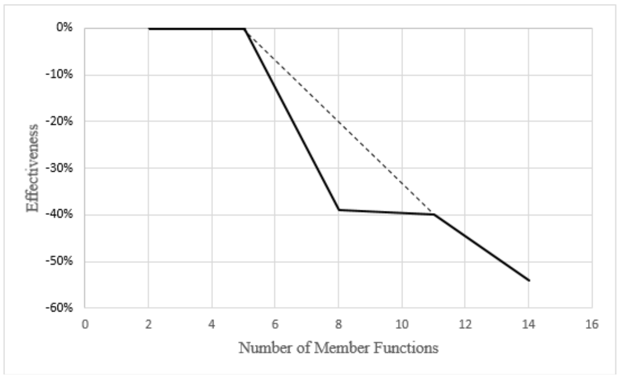
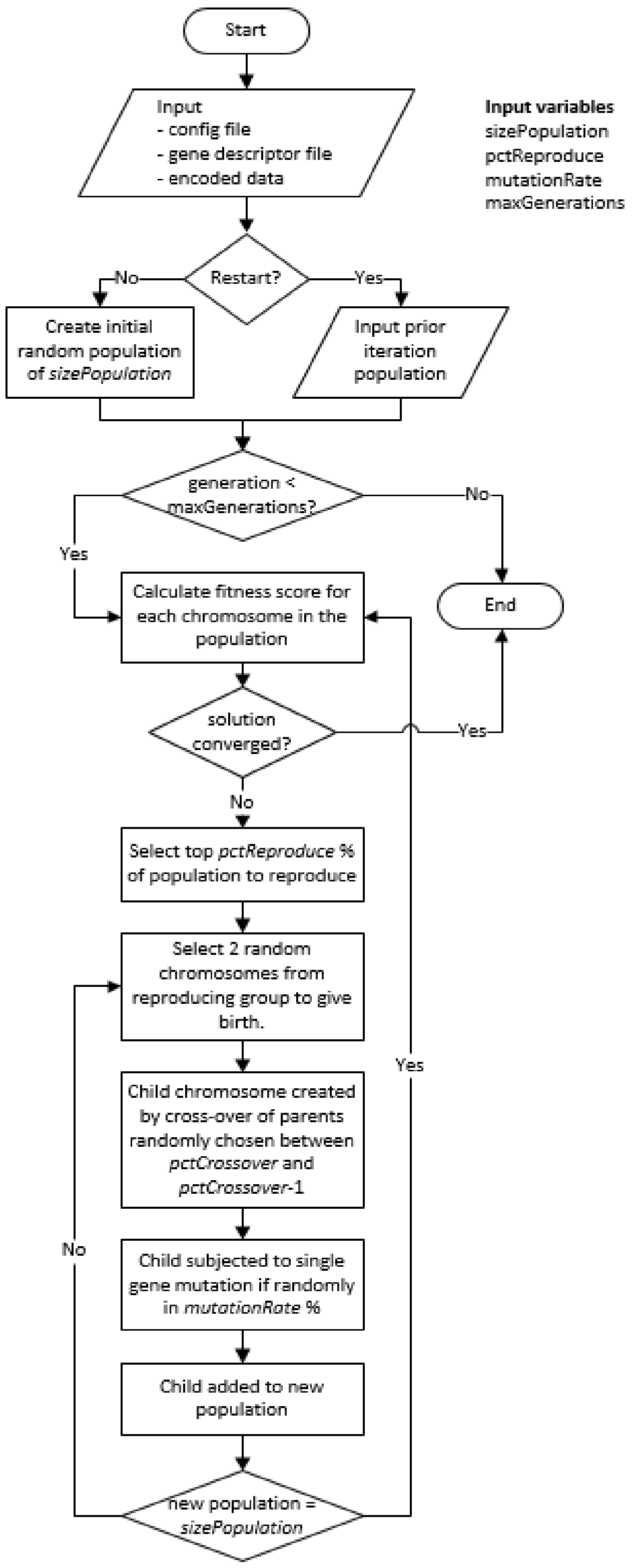


Figure 8: Detection rate sensitivity to fuzzy granularity.

.

Appendix A – Evolve Flowchart


Start
Input
- config file
- gene descriptor file
- encoded data
Input variables
sizePopulation
pctReproduce
mutationRate
maxGenerations
Restart?
No
Yes
Create initial random population of *sizePopulation*
Input prior iteration population
generation < maxGenerations?
Yes
No
End
Calculate fitness score for each chromosome in the population
solution converged?
Yes
No
Select top *pctReproduce* % of population to reproduce
Select 2 random chromosomes from reproducing group to give birth.
Child chromosome created by cross-over of parents randomly chosen between *pctCrossover* and *pctCrossover*-1
Child subjected to single gene mutation if randomly in *mutationRate* %
Child added to new population
new population = *sizePopulation*
No
Yes