CHAPTER 5: EGA TECHNIQUE FOR CLOUD WORM DETECTION

5.1 Introduction

Genetic algorithm (GA) has been used in computer security to solve the security related issues by several researchers (e.g., Goranin & Cenys, 2015; Dhopte *et al.*, 2014; Hoque *et al.*, 2012). GA operates on a population of potential solutions applying the principle of survival of the fittest to produce better and efficient approximations to the solution of the problem that GA is trying to solve. At each generation, a new set of approximation is created by the process of selecting individuals according to their level of fitness value in the problem domain and breeding them together using the operators borrowed from the genetic process performed in nature by crossover and mutation. This process leads to the evolution of populations of individuals that are better adapted to their environment than the individuals that they were created from, just as it happens in natural adaptation.

The functioning of genetic algorithm starts with randomly generated population of individuals. Through various generations, these populations evolved and individuals' quality gets improved. In every generation, there are three basic operators of genetic algorithm; these are: selection, crossover, and mutation, which are applied to each individual. Crossover means exchanging the genes between two chromosomes while mutation means random changing of a value of a randomly chosen gene of a chromosome. These individuals are representation of the problem required to be solved. Different positions of each individual can be encoded as bits, characters and numbers.

In this research, GA was used to improve cloud worm classification as well as for the prediction of future cloud worm attack. The results of this work leads to the improvement of the detection accuracy rate.

There are few works that are based on genetic algorithm for malware detection in cloud computing environment (Kumar & Gohil, 2015; Majeed & Kumar 2014; Li *et al.*, 2012; Goyal & Aggarwal, 2012). These works are focused on malware detection

in cloud using anomaly detection including the use of old dataset for the evaluation. However, this work focused on worm detection in cloud based on dynamic analysis and the latest dataset from virusshare repository.

5.2 Related Works

For classification, six classification algorithms are used, where four of them are in-built in weka in order to identify the most accurate classification algorithm. This work is focused on the prediction of future cloud worm detection. Therefore, a genetic algorithm namely OlexGA proposed by Pietramala *et al.*, (2008) was also integrated. OlexGA is not embedded with weka by default. This work embedded OlexGA in weka and tested in the proposed EGA cloud worm detection technique. This research used four classifiers to make comparison with other works where many other researchers used similar classifier to compare their proposed classifier (Dai *et al.*, 2009; Siddique *et al.*, 2009; Saudi, 2011). The results obtained from classification are discussed and summarised in the following sections.

5.3 Techniques of Genetic Algorithm in Cloud Worm Detection

In this section, a genetic algorithm named EGA has been proposed. From the motivation of OlexGA, this research was aimed to improve GA as classifier. For EGA, there are many parameters that are included to make it more customisable for the researchers. All parameters used in the proposed EGA are described in section 5.4. However, the default value represents the best practice for cloud worm detection. The various techniques improvement of the proposed EGA are shown in Table 5.1.

Table 5.1: Improvement comparison of EGA with OlexGA

Techniques	OlexGA	EGA
Selection	Tournament	Selection Proportional of Fitness
Crossover	Uniform Crossover	Tree Crossover
Mutation	Substitution	Tree Mutation
Evolution	Not included	Evolution Controller

5.3.1 Selection Process

Tournament selection is a method of selecting an individual from a population of individuals in a genetic algorithm. Tournament selection involves running several "tournaments" among a few individuals (or 'chromosomes') chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected. On the other hand, fitness proportionate selection which is also known as roulette wheel selection is a genetic operator used in genetic algorithms for selecting potentially useful solutions for recombination. In fitness proportionate selection, as in all selection methods, the fitness function assigns fitness to possible solutions or chromosomes. This could be imagined as similar to a Roulette wheel in a casino. Usually, a proportion of the wheel is assigned to each of the possible selections based on their fitness value. This could be achieved by dividing the fitness of a selection by the total fitness of all the selections, thereby normalising them to 1. Then, a random selection is made similar to how the roulette wheel is rotated. Selection Proportional of Fitness is presented in Figure 5.1. Using this method, worm characteristics will be selected to generate new possible worm characteristics for the future.

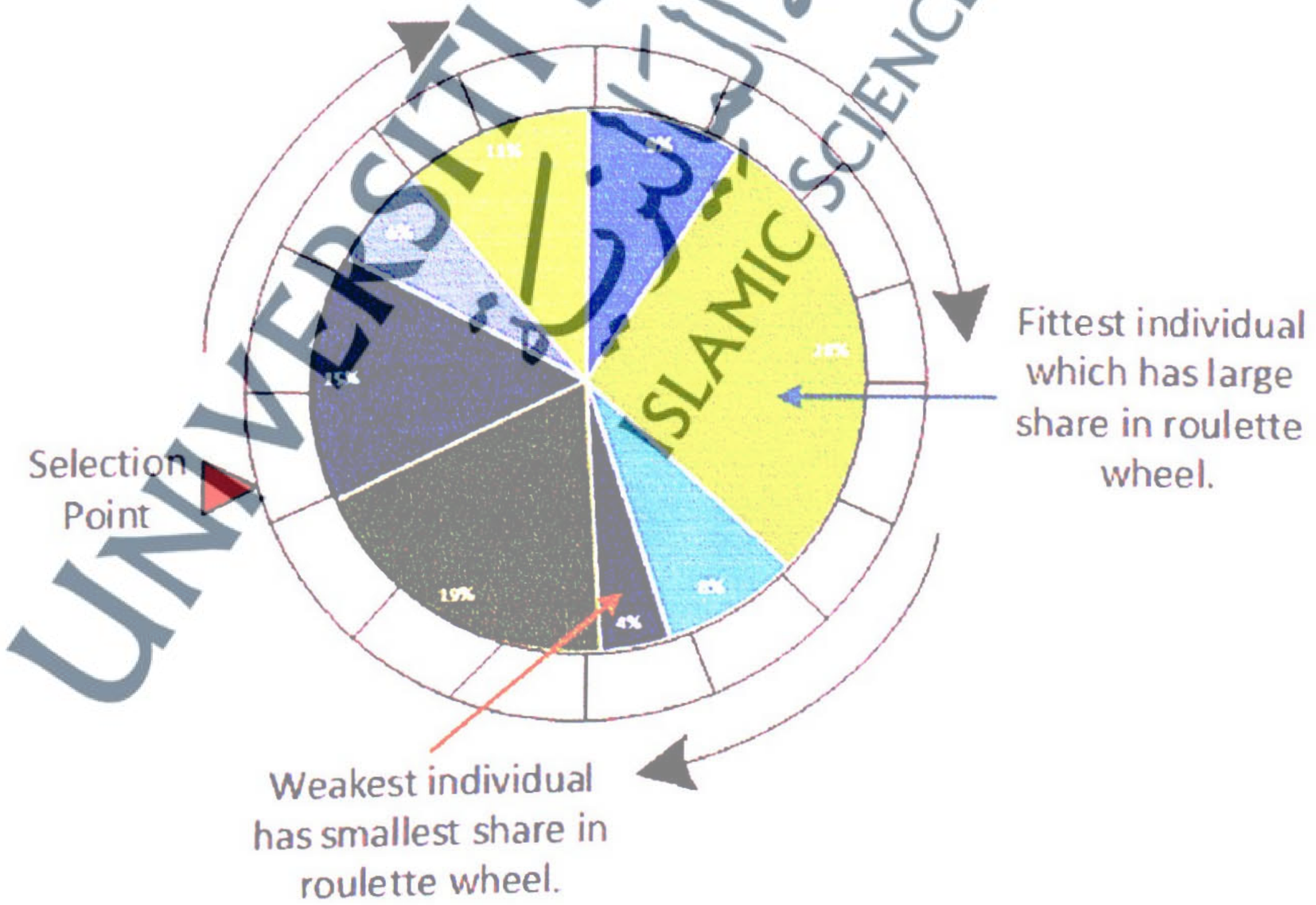

Figure 5.1: Selection Proportional of Fitness

By the combination of the strongest and the weakest worm, new types of worm characteristics could be generated. Through this, it could determine the types of worm which could be developed by the attacker in the future. So, worm detection accuracy will be increased.

5.3.2 Crossover Process

The uniform crossover works in similar way to the multi crossover method, but uses a binary crossover mask with the same length as the chromosome. The parity of the mask bits tells whether to use the information of parent A or B. Using this method, crossing at any point is possible (bit tells the borders not the genes). In tree crossover, given two parse trees as parents, tree crossover selects a sub tree at random from each parent and replaces the selected sub tree in one parent with the selected sub tree from the other. Figure 5.2 shows the tree crossover process.

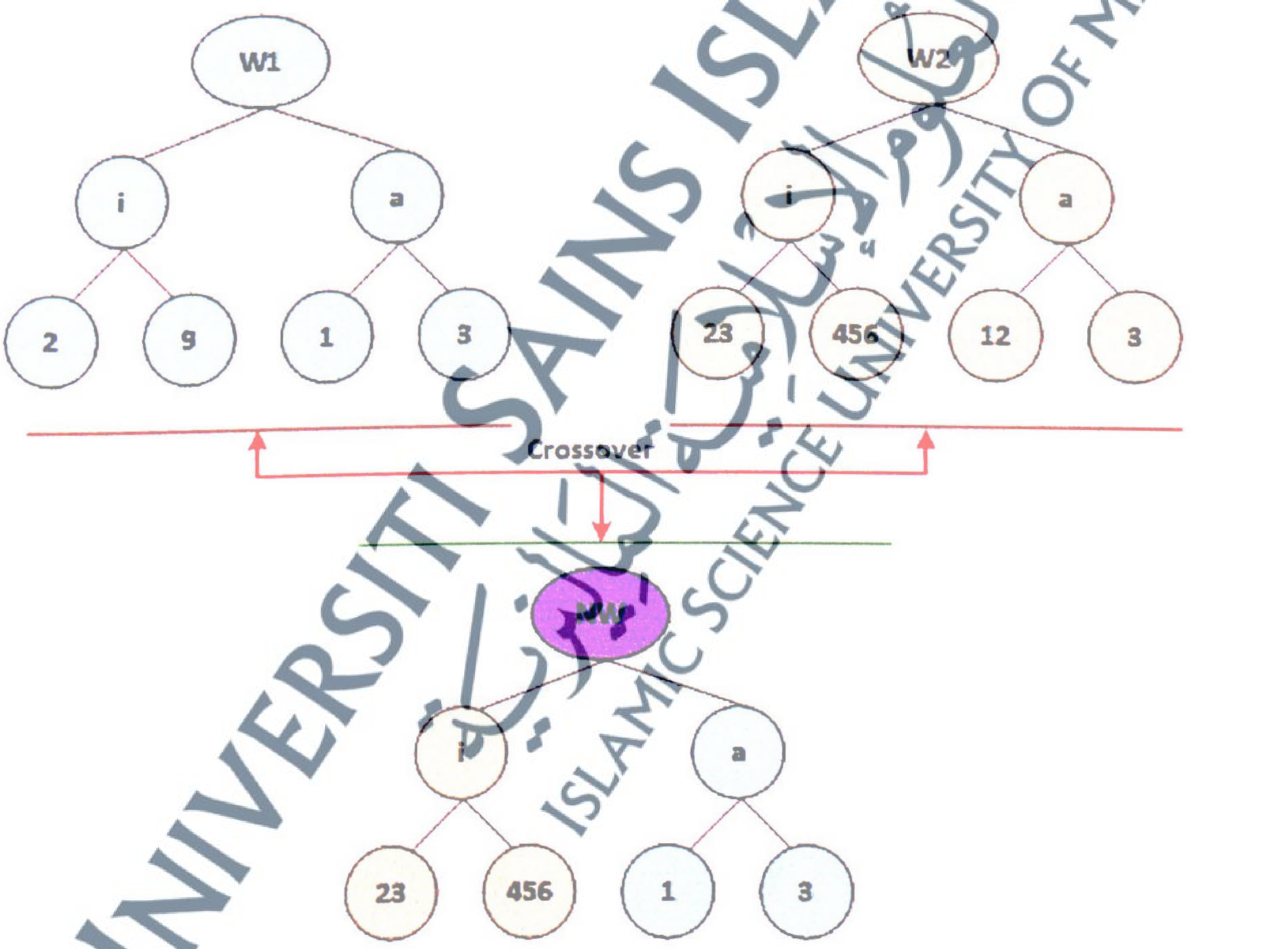

Figure 5.2: Tree crossover

In crossover characteristics, selected worm1 and worm 2 will be combined and will generate tentative types of new worm. In the other words, new worm with the new characteristics will be generated. In each step of GA process, every time new worm is generated with new characteristics. So, detection accuracy will be higher because the system predicts what type of worm could come in the future.

5.3.3 Mutation Process

A generic substitution takes as a key a mapping from each sample to a character. This method is particularly interesting in breaking these ciphers in an automated manner or through unusual means. A novel approach to the problem may reveal something interesting about cryptography in general or if successful, might be applicable to other more secure ciphers. However, a tree mutation operator is used to swap two nodes within a single assembly tree. The two swapped nodes are chosen randomly. This kind of tree mutation introduces new characteristics of worm which is quite different from the substitution. Compared to other type of mutation, tree mutation can give a new worm new and different characteristics which are quite different form the parent's characteristics. However, substitution can give new worm new characteristics that are quite similar to the parent's characteristics. Figure 5.3 shows the tree mutation process.

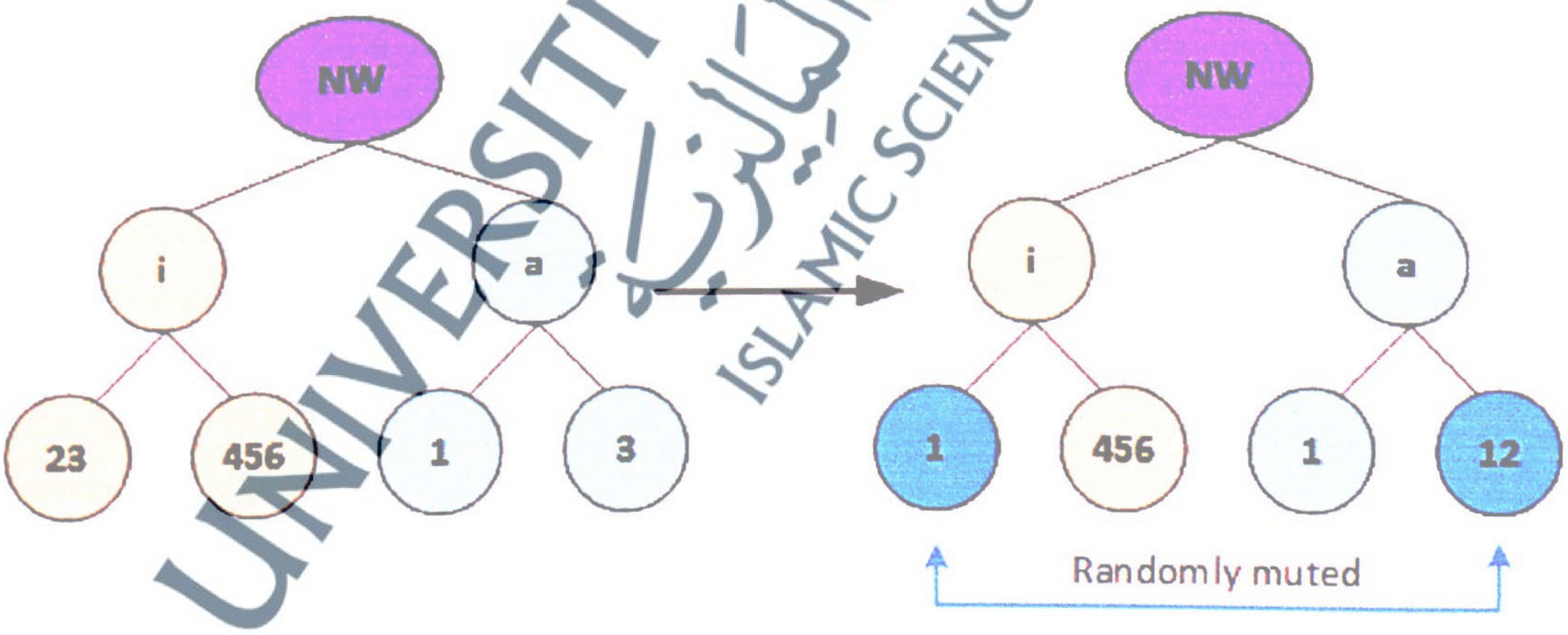

Figure 5.3: Tree mutation

5.3.4 Evolution Process

Evaluation controller controls the evolution process of genetic algorithm. This process ensures better new generation for the future. Figure 5.4 shows how evaluation controller works. Mutation and evolution process is the final steps of GA algorithms. Here, a child worm is mutated and produces better generation by the evolution process. Better generation means worm with good characteristics; it means worms that can survive the new prevention and detection system. If it is possible to predict the characteristic of future worm, it surely will improve detection accuracy.

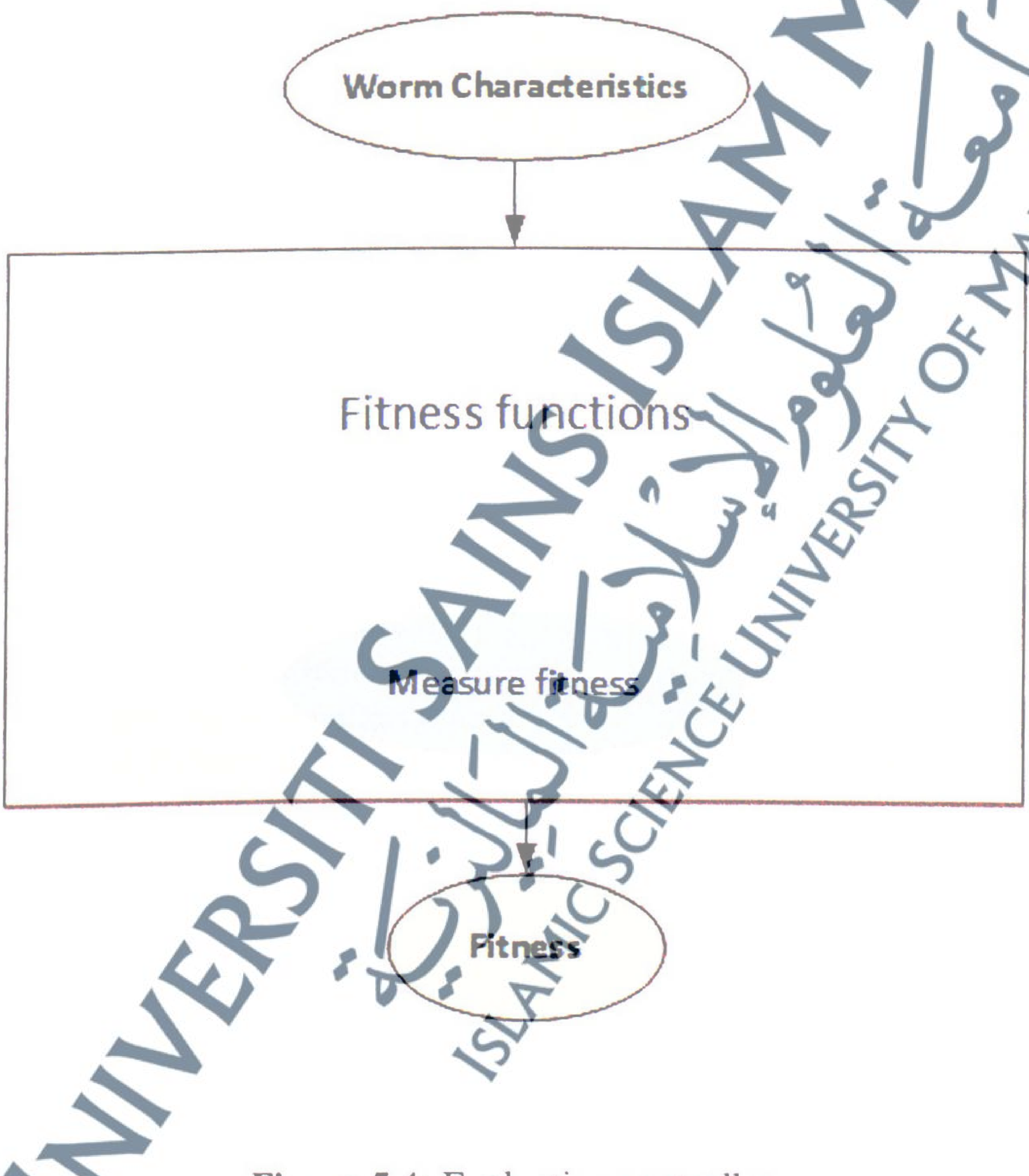

Figure 5.4: Evaluation controller

Table 5.2: Comparison between OlexGA and EGA techniques

Techniques	OlexGA		EGA	
	Technique	**Description**	**Technique**	**Description**
Selection	Tournament	**1**-Selects only the individual with the highest fitness value from certain number of individuals in the population. **2**-Smaller chance to select weak individual. **3**-Loss of diversity. **4**-Does not generate new Worms with new characteristics. **5**-Shows bias with unlimited spread.	Selection Proportional of Fitness	**1**-Selects the strongest and the weakest individual. **2**-Generates new types of worm with new characteristics by the combination of the strongest and the weakest worm which leads to increased worm detection accuracy. **3**-Increases diversity. **4**-Shows no bias with unlimited spread.
Crossover	Uniform Crossover	**1**-Can decrease the quality of parents with good fitness and produce worse offspring. **2**- Creates offspring that is a little bit different from their parents.	Tree crossover	**1**- Increases the quality of parents with good fitness and produces good offspring. **2**-Creates offspring that is very different from their parents. **3**-New worm with new characteristics will be generated.
Mutation	Substitution	**1**-Substitution can give new Worm new characteristics that are quite similar to the parent's characteristics.	Tree Mutation	**1**-Tree mutation can give a new worm new and different characteristics which are quite different form the parent's characteristics.
Evolution	Not included	None	Evolution Controller	1- Controls the evolution process of genetic algorithm. **2**-Ensures better new generation for the future. Better generation means worm with good characteristics. **3**-It is possible to predict the characteristic of future worm; it surely improves detection accuracy. .

Table 5.2 shows the comparison between the OlexGA and EGA techniques. OlexGA has some limitations during it is implementation such as loss of diversity. It also does not generate new worms with new characteristics. In addition, based on all the explanations in the literature review, GA tends to have more strength as compared to other algorithms. For instance, part of the GA's advantages is that it has the capability to learn the malware behaviour. It also helps in predicting malware attacks. It is also noted that GA has reported lower false positive rate as well as higher rate for detecting malware attacks (Nag & Singh, 2015; Majeed & Kumar, 2014; Modi *et al.*, 2013; Binitha & Sathya, 2012; Yusoff & Jantan, 2011). Therefore, this research is focused on genetic algorithm to find better solution to improve cloud worm accuracy detection and cloud worm prevention.

On the other hand, based on previous research as stated in the literature review, OlexGA was implemented in different fields, and has demonstrated higher efficiency and effectiveness in different fields. It is also noted that OlexGA tends to be more stable and powerful. Moreover, its induction method is also more efficient and more accurate as compared to other algorithms (Shivagunde & Kulkarni, 2016; Khedikar & Kulkarni, 2014; Lobo & Rahate, 2013; Manjula, 2011; Rullo *et al.*, 2012). OlexGA has the potential to be utilised for malware detection domain. Nevertheless, it has never been implemented or used for malware detection. Based on the experiment in this research, OlexGA appears to suffer from the use of the tournament selection which causes smaller chances to select weak individual leading to loose individual diversity. Additionally, it also suffers from uniform crossover which creates offspring which is a little bit different form its parents. In relation to mutation, OlexGA uses substitution technique which gives it new characteristics that is quite similar to the parent's characteristics. Another limitation is related the lacking in evolution technique. Therefore, this research aims to improve the state-of-the-art OlexGA by also re-implementing OlexGA itself in order to enhance and also increase the accuracy in the detection of worm within cloud environment with the proposed techniques in this research.

5.3.5 Pseudo code of proposed algorithm.

For the selection process as presented in Pseudo code 1, popSize is the total number of dataset, while output is defined as program Positions. First, it checks whether it has predefined fitness value (or not). Then, it assigns the fitness value to best fitness and checks all data in the dataset repeatedly. After that, it assigns fitness value by considering training fitness; if best fitness is less than fitness, it then assigns current fitness value as best fitness. Again, it checks all popSize through probability and then calculates the probable fitness value for all datasets. If there is no predefined fitness value, then it assigns 0 to best Fitness and checks all data in the dataset. Next, it assigns fitness value by considering training fitness. If best fitness is less than the fitness, then it assigns current fitness value as best fitness. Further, it assigns sum fitness plus fitness value and again checks all popSize through probability. Then, it calculates probable fitness value for all datasets and calls the select program function to select best fitted parents and return for further operation.

```
1 Input: DataSet        Output: Selected Parents
2       If FE.lowerIsBetter() then
3       |       bestFitness ← Double.POSITIVE_INFINITY
4       |       |       for all popSize
5       |       |       |       fitness  ←  ((Program)pop.get(i)).getTrainingFitness()

6       |       |       |       if fitness < bestFitnessthen
7       |       |       |       |       bestFitness ← fitness
8       |       |       |       sumFitness ← sumFItness + fitness
9       |       |       for for all popsize
10      |       |       |       probability[i]  ←  1.0  -
(((Program)pop.get(i)).getTrainingFitness() / sumFitness)
11      else
12      |       bestFitness ← 0
13      |       for all popSize
14      |       |       |       fitness ← ((Program)pop.get(i)).getTrainingFitness()
15      |       |       |       if(fitness > bestFitness)        then
16      |       |       |       |       bestFitness ← fitness
17      |       |       |       sumFitness ← sumFItness + fitness
18      |       for all popsize
19      |       |       probability[i]  ←  ((Program)pop.get(i)).getTrainingFitness()  /
sumFitness
20      | return selectPrograms()
```

Figure 5.5: Pseudo code 1 (Selection)

In crossover process, a vector object is created to perform crossover operation for all new born children requested for crossover. Then, children (two) are selected for crossover and each pair of child is selected to perform cross operation. After that, crossover operation is performed. Here, if there is an odd number of programme, the last one crosses itself when it selects the last child as new born child. Crossover operation is performed, and if replacement of mutated child is necessary, then it will execute the replacement process of muted child and return the crossed child.

```
1       input: new born child, new born parent, probable mutation  Output: muted data
2       Vector resultPrograms ← new Vector()
3       for all nbOfChildren
4       |       resultPrograms.add( ( (ProgramPosition)inputProgramPositions.get(i%
2)).getProgram().clone(PR))
5       for each pair of nbChildren
6       |       ((MainProgramTree)resultPrograms.get(i*2)).crossover(PR,
((MainProgramTree)resultPrograms.get(i*2+1)), fe, trainIns, valIns, pV)
7       if nbOfChildren%2==1 then
8       |       int lastChild ← nbOfChildren – 1
9       |       ((MainProgramTree)resultPrograms.get(lastChild)).crossover(PR,
(MainProgramTree)resultPrograms.get(lastChild), fe, trainIns, valIns, pV)
10      if doReplacement()
11      |       replace(inputProgramPositions, resultPrograms, pop)
12      return resultPrograms
```

Figure 5.6: Pseudo code 2 (Crossover)

For the mutation process as presented in Pseudo code 3, it creates a vector object to perform mutation operation for all new born children requested for mutation. Then, it selects part of child for the mutation of all new born children requested for mutation and performs tree mutation operation. If replacement of mutated child is necessary, then it will do the replacement process of muted child, and finally, it returns the muted child.

```
1       input: new born child, new born parent, probable mutation  Output: muted data
2       Vector resultPrograms ← new Vector()
3       for all nbOfChildren
4       |       resultPrograms.add(
(MainProgramTree)((MainProgramTree)((ProgramPosition)inputProgramPositions.ge
t(0)).getProgram()).clone(PR) )
5       for all nbOfChildren
6       |       ((MainProgramTree)resultPrograms.get(i)).mutation(PR,fe,trainIns,
valIns, pV)
7       if doReplacement()
8       |       replace(inputProgramPositions, resultPrograms, pop)
9       return resultPrograms
```

Figure 5.7: Pseudo code 3 (Mutation)

Fitness evaluators are used to evaluate the fitness of a program over a group of instances, and these entail the combination of several functions. For measure Fitness function, it returns fitness of in (single data in a dataset) and prog (Program for fitness evaluation). IsNewchild returns true if this child is new and was not previously present in the samples. Meanwhile, GetSuperiorBound gives the highest value that this fitness evaluator can return, and when evaluating the fitness, it can return an infinite value and can return maximum fitness. Newchild is Better returns true if this classifier's given fitness is better for the newchildren. Is Nominal returns true if the new children are real and can be used as a nominal. IsNumeric returns true if the new children are real and can be used as a numeric.

```
1       input: ins, prog        Output: fitness
2       isNominal()
3       isNumeric()
4       is Newchild()
5       measureFitness (Instances ins, Program prog)
6       getSuperiorBound()
7       Newchild isBetter()
```

Figure 5.8: Pseudo code 4 (Evolution Controller)

Using selection proportional of fitness, tree mutation, tree crossover and evaluation controller; accuracy rate becomes higher. Different methods were used for selection, mutation and crossover compared to OlexGA. These methods improved the accuracy

of cloud worm detection. The results of proposed algorithm are presented in next section.

Pseudo code 1, 2, 3 and 4 are the main functions of genetic algorithm. Pseudo code 1 selects parent from all cloud worms. Then, pseudo code 2 does the crossover operation to produce new child worm. New born child worm then goes into the mutation process by pseudo code 3. Finally, evaluation controller ensures better generation which means it can survive the changing environment of upcoming cloud worm detection and prevention system. In this way, detection accuracy will be increased in the proposed enhanced genetic algorithm.

5.3.6 Application of Genetic Algorithm in Cloud Worm Detection

The proposed GA based approach consists of pre-processing and learning phase. In pre-processing phase, dataset was collected and converted into numerical and normalised form. These steps were done as described in section 3.3.6.1 and section 3.3.6.4. The same dataset was used for all works in this thesis, and therefore, pre-processing was achieved in earlier chapter. The next phase is known as the learning phase. In this phase, GA is used to classify the prepared dataset. GA also produces rules for new generations for the upcoming possible worm threat in cloud. Figure 5.9 shows the basic working principle of GA in cloud worm detection.

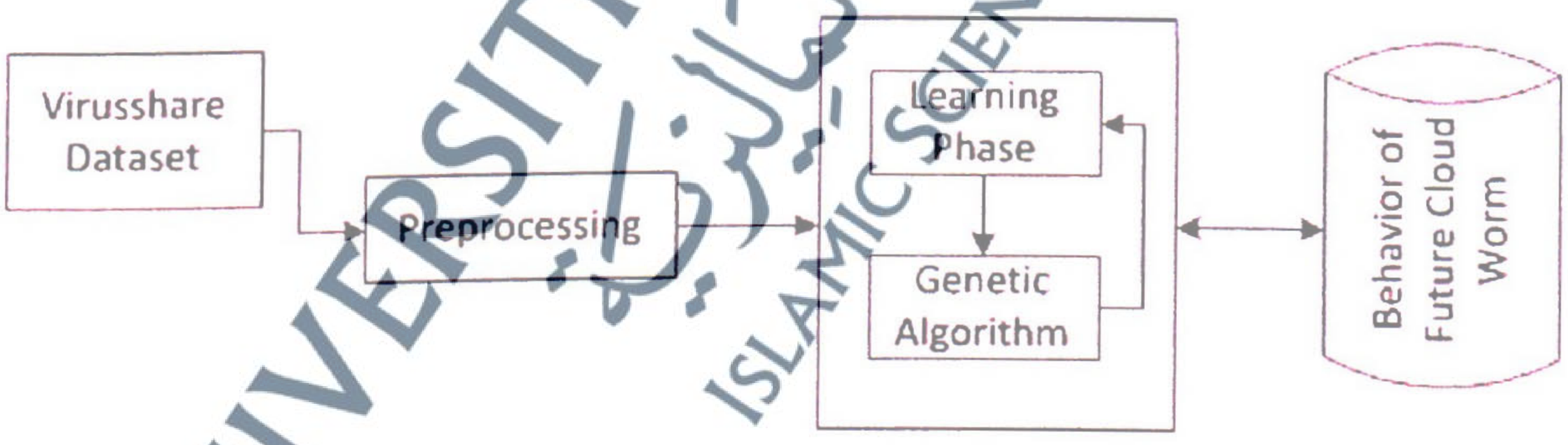

Figure 5.9: GA working flow in cloud worm detection

Steps for pre-processing are as follows:

Step 1: Select classification of Virusshare dataset

Step 2: Convert obtained classification to nominal data

Step 3: Input dataset to WEKA

Step 4: Inclusion of security metric in learning phase

Operations of genetic algorithm are explained as follows:

Step 1: Take an input from the pre-processing unit

Step 2: Select random two parents for crossover

Step 3: Apply mutation operator on selected parents

Step 4: Check whether newly generated worm behaviours are already in database

Step 5: If generated worm behaviours are new then check the fitness value

Step 6: Store if the fitness value is better

During the research process of cloud worm detection using genetic algorithm, various methods were used for crossover, selection and mutation. In the pre-processing phase, classification was identified by dynamic analysis of Virusshare dataset. This classification could be passed to Weka unless it transformed these classifications into nominal data. Due to this, the classification results are transformed into nominal data. Then, security metric is included for the learning phase.

The proposed genetic algorithm takes an input from the pre-processing unit. Then, it selects random two parents for crossover. After that, mutation operator is applied on selected parents. Next, it checks whether newly generated worm behaviours are already in database (or not). If yes, then it will repeat the iteration. Otherwise, generated worm behaviours are checked for the fitness value. Finally, it stores it as better fitness value if fitness value is better. The flowchart of the EGA experiment can be seen in Figure 5.10 below. This flowchart describes the detailed steps of developing EGA for cloud worm detection.

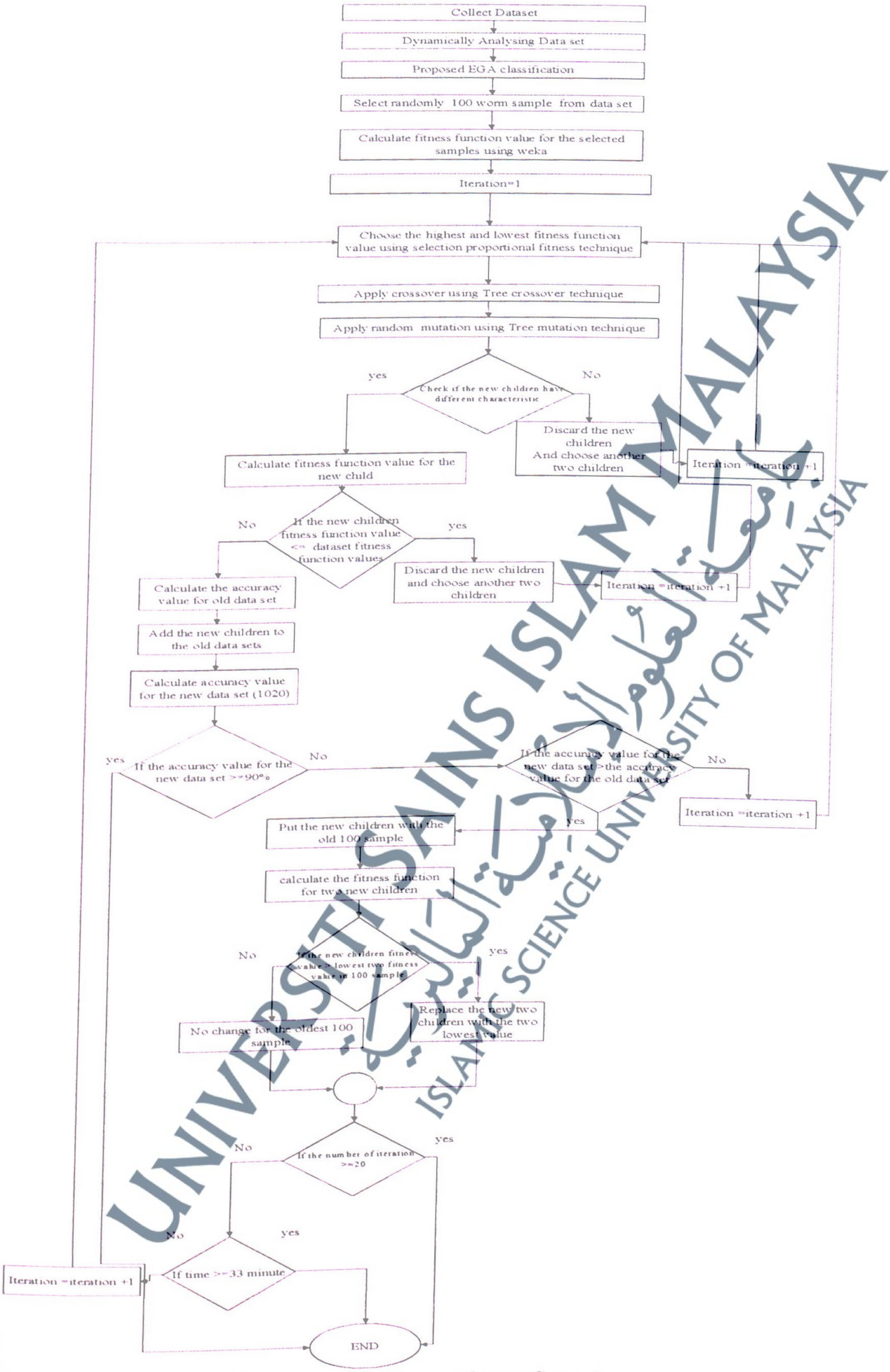

Figure 5.10: EGA experiment flow char

5.4 Parameters of Proposed Genetic Algorithm

Several parameters are implemented and tested in the proposed genetic algorithm. The functionality of these parameters is described below.

5.4.1 biasConstant

Bias constant parameter is used to control the selection process. Biasing could be 0% to 100%, and thus, this parameter passes by 0 to 1. 0 means it does not use biasing during the selection process. On the other hand, if the value is 1, it will be biased 100% selection process. Accordingly, 50% biasing value which is 0.5 is performed best as bias constant. The control for adding new children in population is based on their fitness function, and if the fitness function of the new child is >= 50 %, it will add the new children into the population. There is no need for children within the low fitness as it will lead to low accuracy rate.

5.4.2 evolutionController

Evolution controller parameter chooses when and how many children will be produced to replace the preceding generation. It produces a number of children and chooses between children and parents to create a new generation.

5.4.3 fitnessEvaluator

Three options are available for fitness evaluation, Hybrid classifier with booting was chosen because of its hybrid nature for evaluating the fitness. The available options for fitness evaluation are: i) Confidence on one class recognition (hybrid classifier with boosting). ii) Error sum (continuous), and iii) Root mean square error (continuous). Other two options are based on the calculation or error of the fitness evaluator and it works continuously throughout the evaluation process.

5.4.4 newPopulationSize

This parameter represents the size of children population which is 100 percent in the proposed algorithm case. It selects all children after running the method. All children population are considered as new population.

5.4.5 operatorChildren

The children operator for all options mutation, mutation (node only), reproduction, and new program are one. Nevertheless, the operator of crossover is two which denotes that the crossover will be done two times in each process.

5.4.6 operatorParents

Parents operator for all options mutation, mutation (node only), reproduction, new program is one. Nevertheless, the operator of crossover is two which denotes that the crossover will be done two times in each process.

5.4.7 populationInitializer

Population initialiser selects the method for initialising a population for programs. For the experiment, tree crossover method was used for its best performance. On the other hand, this parameter performs tree crossover operation.

5.4.8 populationSize

Population size represents the number of programs in the population. The size of the population is 100 in the proposed genetic algorithm. This is because all populations are used throughout the process.

5.4.9 programSelector

Program selector selects the program method for the reproduction process. This parameter is used to perform the selection proportional to fitness technique.

5.4.10 Completion

Completion parameter controls the fitness function, and the three parameters could be passed through this parameter. The first parameter defines target fitness; if it is not 90%, then it means it does not perform better. The next parameter defines the maximum number of generation which is 20, and the maximum total time can be taken by the process which is 0.33 minutes. However, computation time varies from system to system but it should not take more than the defined minutes. This time parameter is fixed because if the system took more time than that, it means something is either wrong with the system or it could be a worm attack.

5.5 Improvement in proposed EGA and reason for choosing GA

There are many improvements that have been done in proposed EGA compared to OlexGA. Several new parameters proposed in EGA also include all other processes that are completely different compared to OlexGA. Subsection 5.5.1 describes how the parameters are different in EGA compared with OlexGA. The next section which is subsection 5.5.2 describes the processing parts of EGA which makes it more advanced and efficient than OlexGA. Further, subsection 5.5.3 describes why GA is chosen among other algorithms.

5.5.1 Differences between EGA and OlexGA parameters

EGA used completely different methods in each step of the processing compared to OlexGA. There are also 10 new parameters added which makes EGA more enhanced than OlexGA. The parameters of biasConstant, evolutionController, fitnessEvaluator, operatorChildren, operatorPatrents and PopulationInitializer are completely new which differentiate EGA from OlexGA. In particular, OlexGA does not use PopulationInitializer. Secondly, OlexGA only depends on crossover and mutation process but EGA uses various operators in parents, children and proportions during the whole process. Finally, EGA has evolutionController and fitnessEvaluator which are missing in OlexGA. Additionally, the methods used inside those parameters of EGA are completely different compared to OlexGA. A detail explanation of each parameter is explained in section 5.4.

5.5.2 Part of EGA that improves OlexGA

OlexGA follows three basic processes of genetic algorithm. They are: selection, mutation and crossover. On the other hand, EGA added another process which is called new program process and evolution. These new program processes are based on the past experience of the algorithm runtime which is controlled by the evolutionController parameter. Moreover, the basic four processes used in EGA are completely different from those of OlexGA. For selection process, OlexGA used Tournament selection but EGA used proportional of fitness function. Further, substitution mutation is used in OlexGA. On the other hand, EGA used tree mutation. OlexGA used uniform crossover for the crossover process while EGA used tree crossover. EGA used evolution controller which is not included in OlexGA. All of the above processes are described in section 5.3 in details.

5.5.3 Reason for choosing GA over other algorithms

GA system works based on the codes that are prevalent in the scope of the identified problems, compared to the traditional systems that work on the problem scope itself. Besides that, GA systems provide a higher number of effective solutions compared to the conventional method of producing only one method to solve a problem. Therefore, GA system is more efficient. On the other hand, conventional systems employ the use of derivatives to evaluate a solution, whereby GA system employs the use of a fitness function to evaluate the highly optimal solution that it has produced. Another fact that genetic programming had reported is a lower false positive rate as well as a higher rate for detection of malicious attacks (Lu & Traore, 2004). The performance of proposed EGA is compared with other classifier such as J48, IBK and Naïve Bayes. EGA showed better performance compared to other algorithms which are described in details in section 5.7.

Some works used genetic algorithm-based malware detection in cloud computing environment (Kumar & Gohil, 2015; Majeed & Kumar 2014; Li *et al.*, 2012; Goyal & Aggarwal, 2012). These works were focused on malware detection in cloud using anomaly detection and were not based on weka. They were also using old dataset for their evaluation. To compare the performance of the proposed EGA, it is necessary to

compare with the performance of another GA classifier, where OlexGA is also based on GA as a classifier that is available which is revealed from the related literatures as presented in section 2.11. Due to this, OlexGA was chosen to verify the performance of the proposed EGA classifier. The evaluation was executed through weka for this research.

5.6 EGA Improvement and Evaluation Over OlexGA.

In this section, the improvement evaluation of EGA over OlexGA is explained

A. OlexGA Technique

OlexGA, together with the corresponding selection, crossover and mutation techniques were selected as shown in Table 5.3 below:

Table 5.3: OlexGA Techniques

Selection	Crossover	Mutation
Tournament	Uniform Crossover	Substitution

Figure 5.11 below shows the initial parameters related to OlexGA techniques.

Figure 5.11: OlexGA Techniques and Parameters

Figure 5.12 below shows the results for OlexGA algorithm:

```
=== Run information ===

Scheme:weka.classifiers.rules.olexGA -Y 0 -P 60 -X 0 -R 1.0 -M 0.001 -S 1 -I 500 -G
200 -A 1 -E 0.2 -C 1
Relation:     Classification_result
Instances:    1195
Attributes:   6
              Infection
              Activation
              Payload
              Operation algorithm
              Propagation
              worm
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         845               70.7113 %
Incorrectly Classified Instances       350               29.2887 %
Kappa statistic                          0.3612
Mean absolute error                      0.2929
Root mean squared error                  0.5412
Relative absolute error                115.8487 %
Root relative squared error            152.3533 %
Total Number of Instances              1195

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.656     0          1         0.656     0.792      0.828    worm
                 1         0.344      0.336     1         0.503      0.828    Benign
Weighted Avg.    0.707     0.051      0.902     0.707     0.75       0.828

=== Confusion Matrix ===

   a   b   <-- classified as
 668 350 |   a = worm
   0 177 |   b = Benign
```

Figure 5.12: Results of OlexGA algorithm

As can be seen in Figure 5.12 above, the output shows the result for OlexGA. The accuracy rate is 70.71%; the True Positive result is shown as only 70.7%. As for the False Positive rate, the performance of OlexGA is 5.1%; the Precision rate is 90.2% and the Recall rate is 70.7%. Finally, the output for the F-Measure rate is shown to have the value of 7.5%. The result indicates the accuracy rate; True Positive, Recall and F-Measure rates are low in comparison to EGA as explained in the following section.

B. EGA Technique

To evaluate the performance of the selected technique for EGA, this technique with the corresponding selection, crossover, mutation and evolution controller as presented in the table below:

EGA, together with the corresponding selection, crossover, mutation and evolution controller techniques were selected as shown in Table 5.4 below:

Table 5.4: EGA Techniques

Selection	Crossover	Mutation	Evolution
Selection Proportional of Fitness.	Tree Crossover	Tree Mutation	Evolution Controller

Figure 5.13 below shows the initial parameters related to EGA techniques.

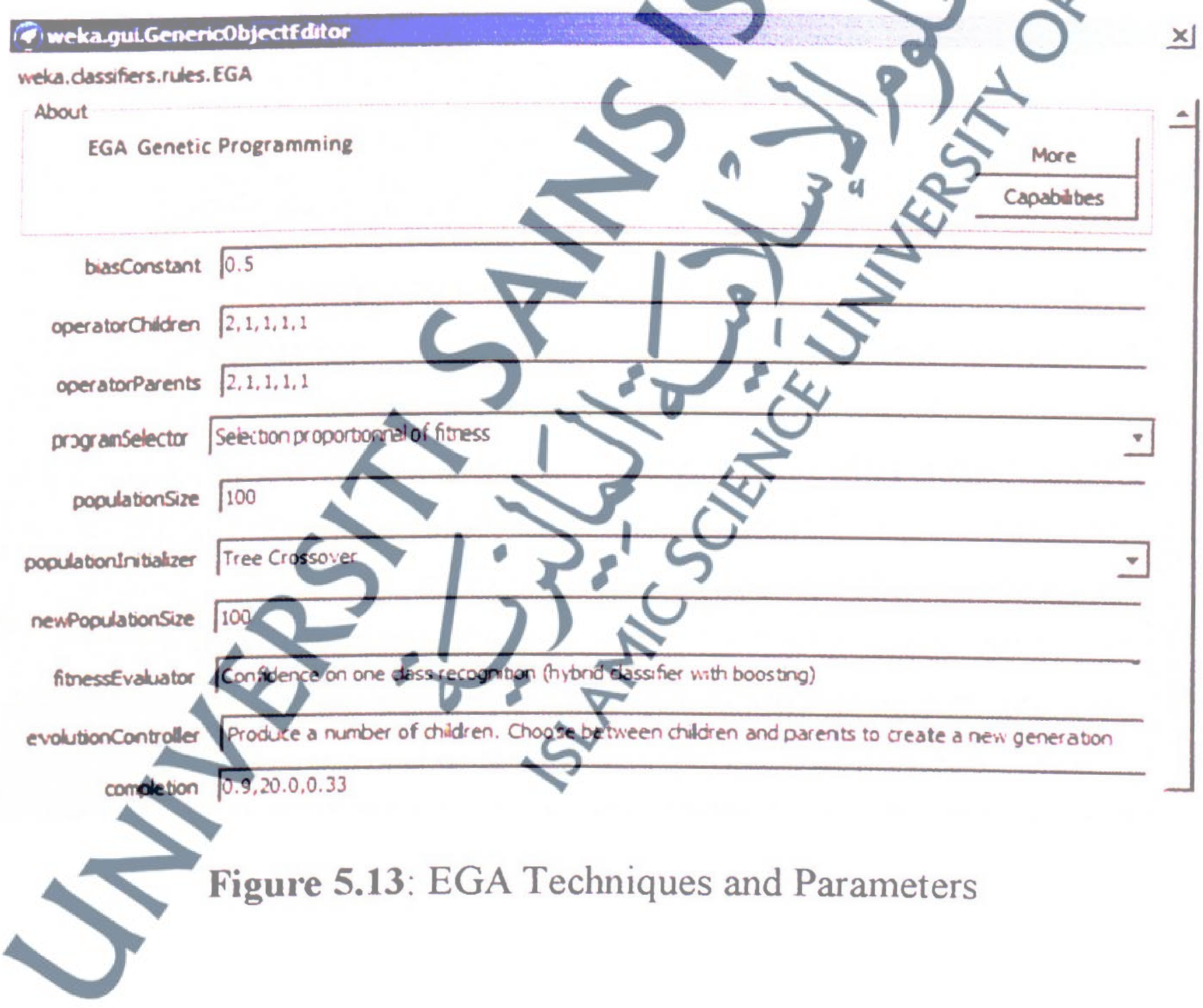

Figure 5.13: EGA Techniques and Parameters

Figure 5.14 below shows the results of EGA algorithm:

```
=== Run information ===

Scheme:weka.classifiers.rules.EGA
Relation:     Classification_result
Instances:    1195
Attributes:   6
              Infection
              Activation
              Payload
              Operation algorithm
              Propagation
              worm
Test mode:10-fold cross-validation

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1192               99.749  %
Incorrectly Classified Instances         3                0.251  %
Kappa statistic                          0.99
Mean absolute error                      0.0025
Root mean squared error                  0.0501
Relative absolute error                  0.993  %
Root relative squared error             14.1052 %
Total Number of Instances             1195

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0.017      0.997      1         0.999      0.992    worm
                 0.983     0          1          0.983     0.991      0.992    Benign
Weighted Avg.    0.997     0.014      0.997      0.997     0.997      0.992

=== Confusion Matrix ===

    a    b   <-- classified as
 1018    0 |    a = worm
    3  174 |    b = Benign
```

Figure 5.14: Results of EGA algorithm

Figure 5.14 above highlights the result of EGA algorithm. The accuracy rate is 99.749%; the True Positive result is shown as 99.7%. As for the False Positive rate, the performance of OlexGA is 1.4%; the Precision rate is 99.7% and the Recall rate is 99.7%. Finally, the output for the F-Measure rate is shown to have the value of 99.7%. The result indicates the accuracy rate; True Positive, Recall and F-Measure rates are higher in comparison to OlexGA. In addition, False Positive rate is lower in comparison to OlexGA which indicates better results.

To conclude, all the output values related to the proposed EGA as explained above have provided a more favourable result compared to OlexGA. This has also proven that the enhancement carried out on OlexGA to improve cloud worm detection have generated a much positive result making it more efficient for cloud worm detection.

Table 5.5 demonstrates the overall experimental results comparing EGA with OlexGA and other algorithms.

5.7 Experimental Results and Comparison for EGA Detection Technique

An experiment was conducted using 10-fold cross validation dataset generated using weka. 10-fold cross validation is the standard way of measuring the error rate of a learning scheme on a particular dataset. For reliable results, the experiment was performed 10 times for the ten-fold cross-validation (Bouckaert *et al.*, 2015). Moreover, 10 fold cross validation divides dataset into 10 parts (folds), then holds out each part in turn, and finally, gives average of the results. So, each data point in the dataset is used 9 times for training and once for testing. By default, weka does cross validation using 10 fold because of comprehensive tests on many different datasets, where different learning techniques have shown that 10 is about the right number of folds to get the best estimate of error (Witten *et al.*, 2005). There are two main reasons that inspire the usage of this type of 10 fold cross validation test. Firstly, it uses as much data as possible for training and testing and secondly, the better accuracy of its findings (Saudi, 2011). Furthermore, the assessment for the purpose of EGA has been evaluated using Weka which is also a Java-based tool that has been used by many researchers for detection, clustering, classification and many more. Researchers that used weka in their study include Bhandari (2015), and Thakur and Mahan (2015).

EGA experimental results were compared with those of OlexGA (Pietramala *et al.*, 2008) and NB tree plus random forest (Bhat *et al.*, 2013). Comparison with other established algorithm is also considered. Various researchers (e.g., Siddiqui, 2008; Saudi, 2011; Bhat et *al.*, 2013; Shahzad, 2014) also used these algorithms for malware detection.

EGA was developed in Java and then integrated with Weka. There was a significant improvement found compared with OlexGA while classifying with EGA. Table 5.5 shows the comparison of the experimental result with those of various algorithms. The EGA Technique for Worm Detection results are incorporated in Appendix C.

5.7.1 True Positive rate Results Analysis

True positive (TP) is the number of predicted cloud worm samples correctly classified and identified as malicious. TP was calculated by using Equation (3.4) as shown in section 3.3.8. EGA shows highest number of true positive which is 99.7 % followed by IBK with 99.2 %. These algorithms outperformed in the case of true positive rate. However, OlexGA shows the lowest true positive which is only 70.7 % and NB Tree + Random Forest showed 99 % TP rate in their work. For J48 and Naïve Bayes, they also performed well by producing 98.9 % and 99.1% TP rate respectively. Figure 5.15 shows the results of TP rate by various algorithms in Weka data mining.

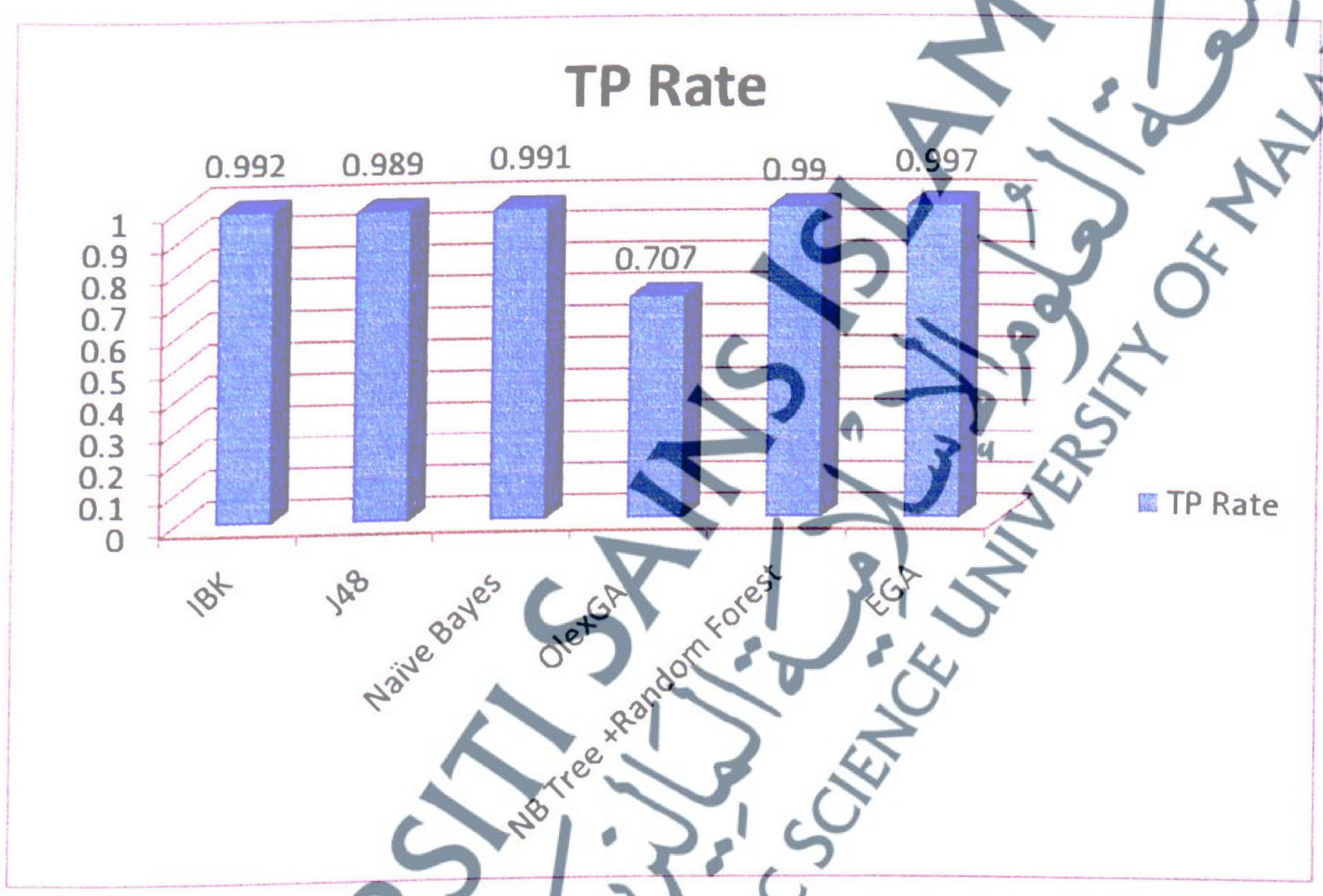

Figure 5.15: TP rate of various classification algorithms

According to Figure 5.15, OlexGA has lower TP rate compared with other established classifiers. However, EGA showed improved TP rate in cloud worm detection using GA and GA used in OlexGA.

5.7.2 False Positive rate Results Analysis

Cloud worm detection accuracy also refers to the correct classification. Then again, the false positive (FP) denotes that the data misclassified it, which means that the data actually belongs to a different class. FN takes place when the data is wrongly classified as a different class, but actually it belongs to another class. In summary, FP is the number of predicted cloud worm samples incorrectly classified as malicious. FP is calculated using the Equation (3.5) as presented in section 3.3.8. In the case of FP, EGA shows the lowest FP rate compared with IBK, J48 and Naïve Bayes. The performance of OlexGA is shown to be worse as the value of FP is 5.1% as shown in Figure 5.16.

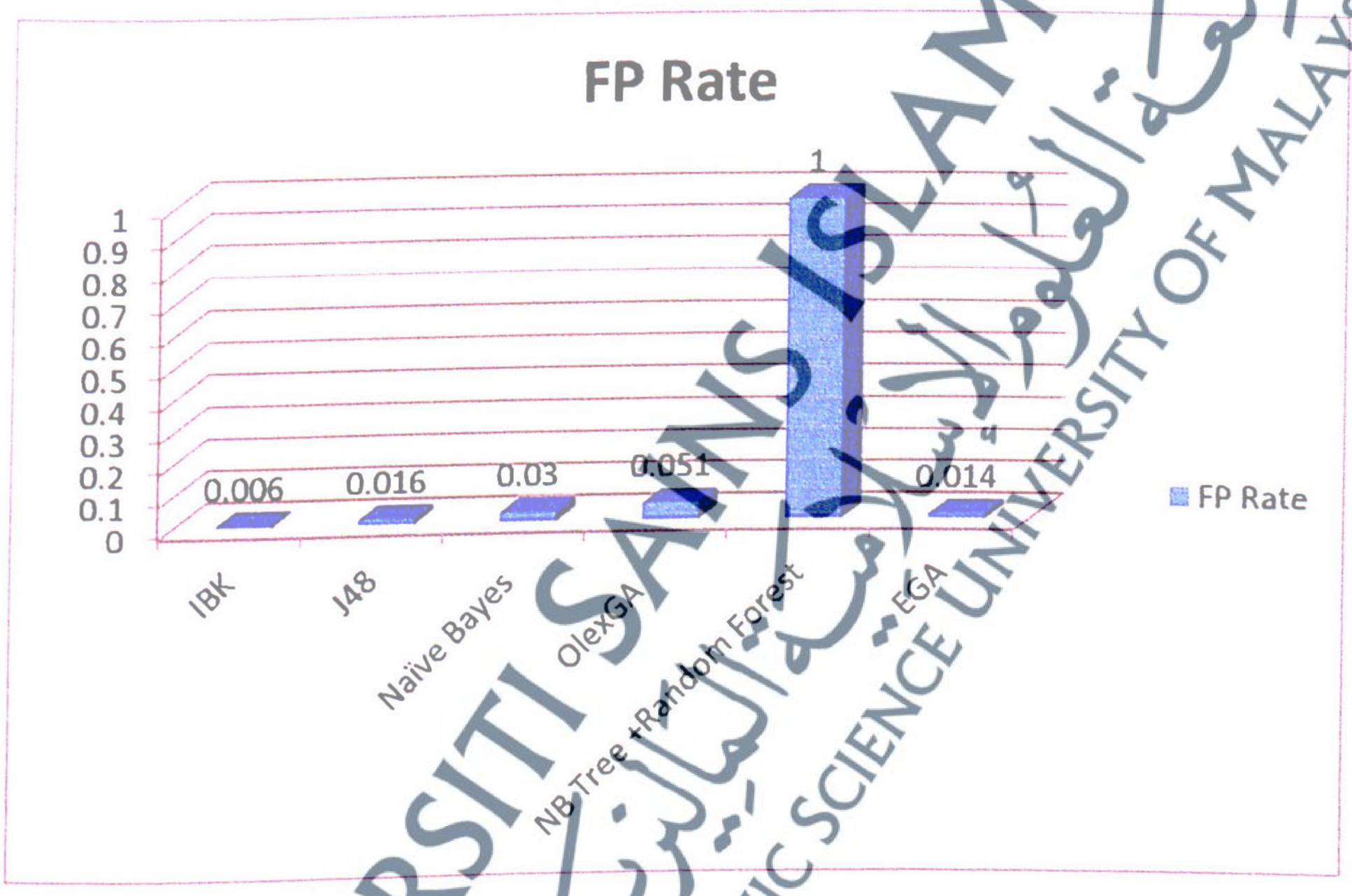

Figure 5.16: FP rate of various classification algorithms

NB Tree + Random Forest showed the highest FP rate. Due to the low TP rate, FP rate is higher than that of OlexGA compared with IBK, J48, Naïve Bayer, and EGA. This work contributes by lowering the FP rate using GA for cloud worm detection.

5.7.3 Precision and Recall rate Results Analysis

Precision explains the proportion of true positive classifications in all positive results. This denotes how many sample were classified correctly and not false positive. Precision is measured by using TP and FP rate as shown in Equation (3.9) as presented in section 3.3.8. If TP is higher, it means higher precision. Due to the fact that EGA incurred good precision value and the value is 99.7% as presented in Figure 5.17. Compared with OlexGA, EGA has 99.7% higher precision rate. Consequences of OlexGA has 90.2% precision due to its low TP and high FP rate.

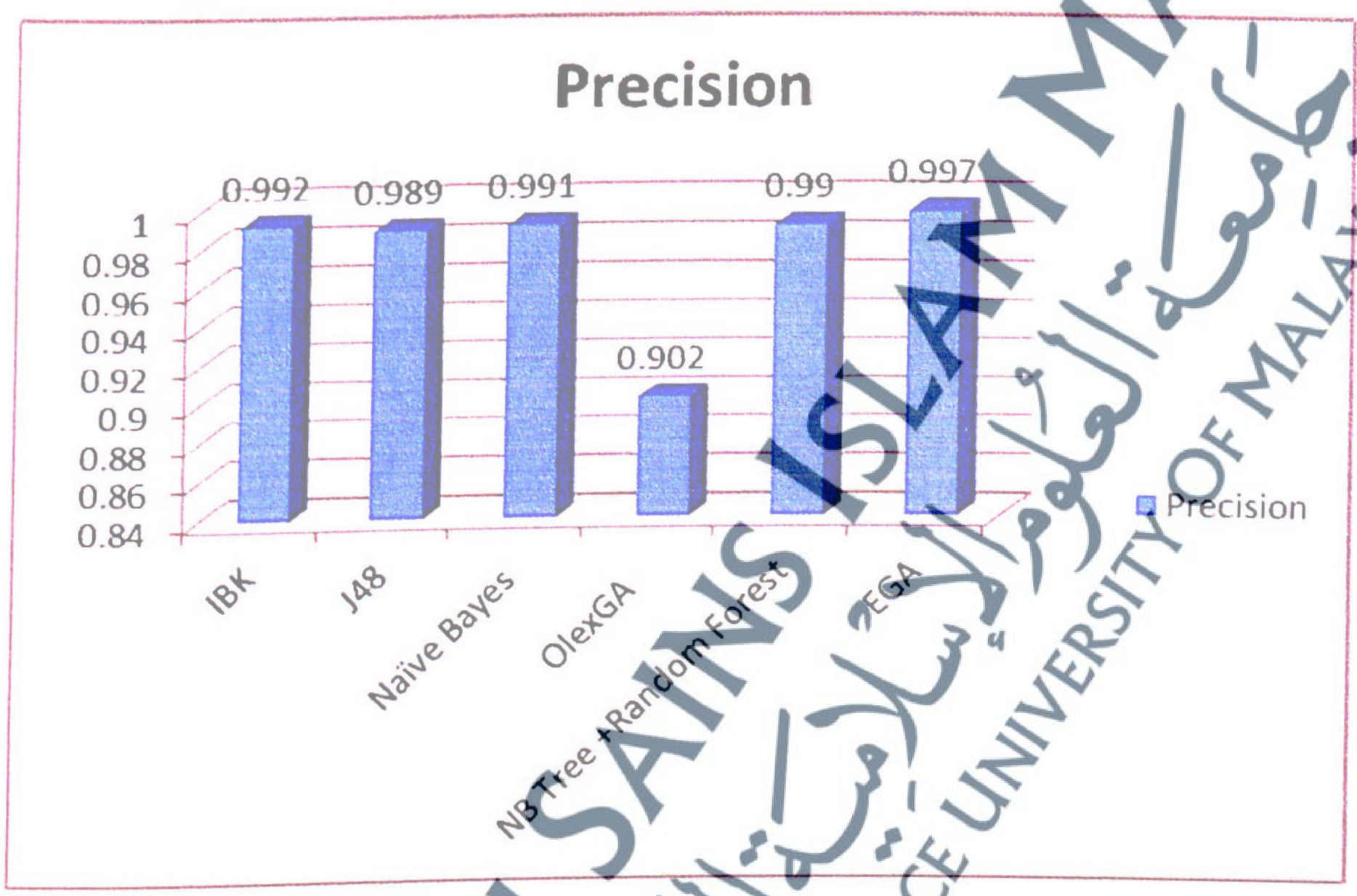

Figure 5.17: Precision rate of various classification algorithms

EGA recall rate is 99.7%. In comparison to OlexGA and NB Tree+ Random Forest, EGA has higher recall rate of OlexGA which has 70.2% recall rate. Similarly, EGA has higher recall rate of NB Tree+ Random Forest which has 99% recall rate. The experimental results of recall rate are shown in Figure 5.18.

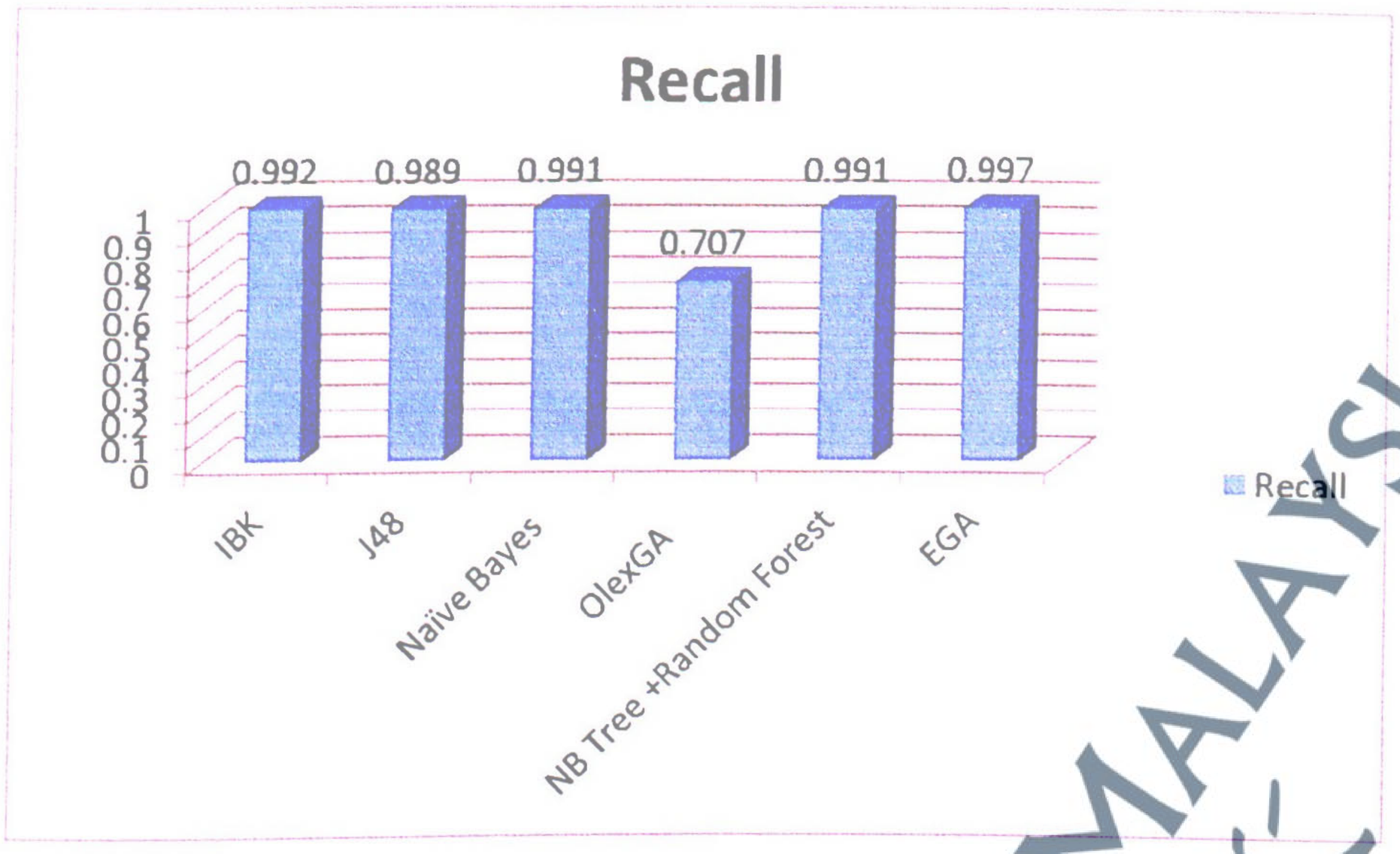

Figure 5.18: Recall rate of various classification algorithms

5.7.4 F-Measure Results Analysis

F-measure is also referred as false negative rate, and the equation of f-measure is shown in Equation (3.8) as presented in section 3.3.8. F-measure entails data points that are classified as malicious but are actually normal. Except for OlexGA, all other classification algorithms performed better as shown in Figure 5.19. However, EGA shows better F-Measure compared to OlexGA. Further, compared with NB Tree + Random Forest and OlexGA, EGA has the highest F-Measure rate.

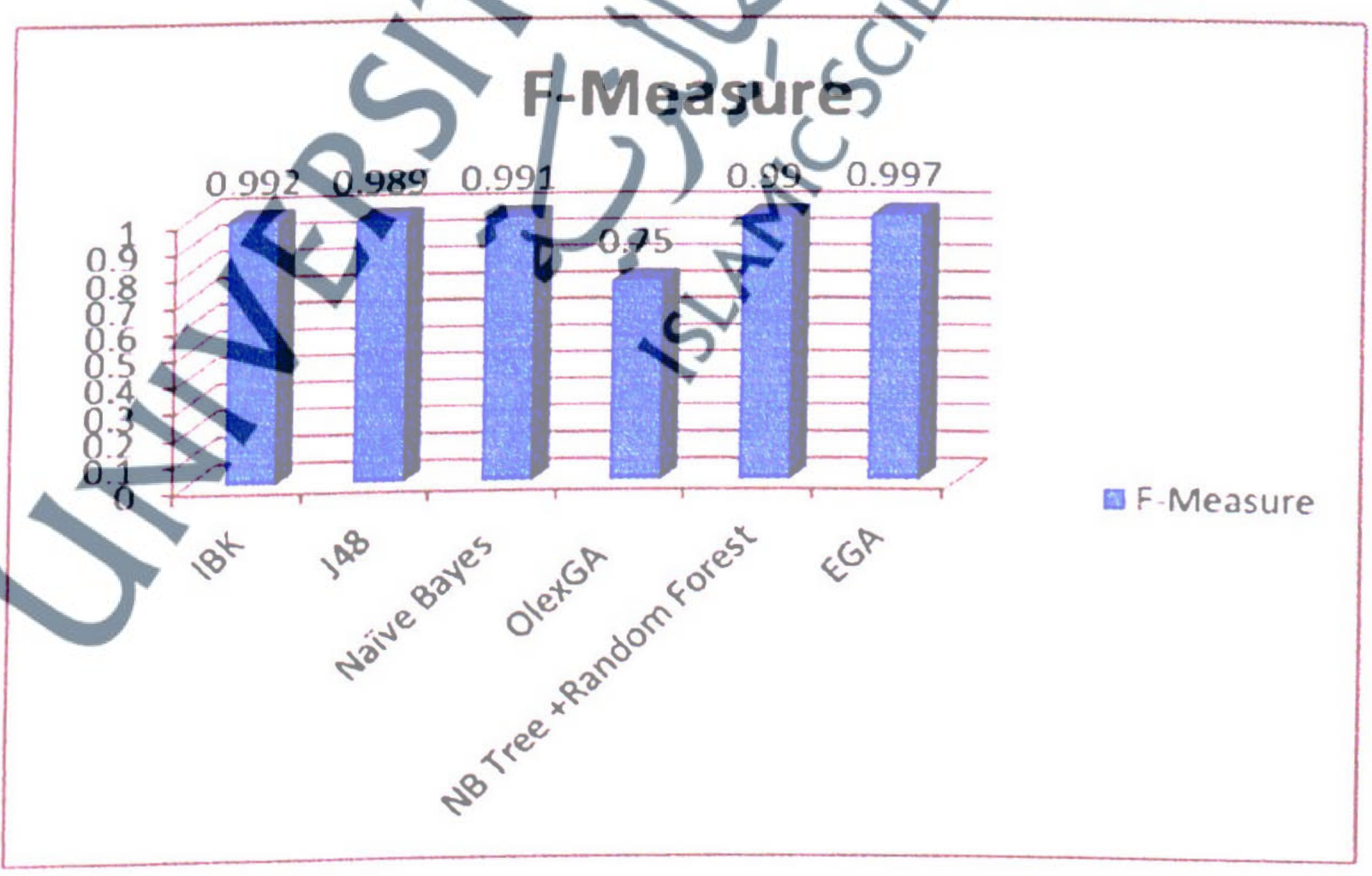

Figure 5.19: F-measure rate of various classification algorithms

From Figures 5.15, 5.16, 5.17, 5.18 and 5.19, it is clearly seen that the proposed algorithm outperformed OlexGA in the case of TP, FP, Precision, Recall, F-Measure and ROC area rate. The proposed algorithm is also better as compared with NB tree plus random forest.

5.7.5 Correctly Classification Results Analysis

Correctly classification is also referred as accuracy. The accuracy rate of various algorithms is illustrated in Figure 5.20. EGA shows the highest number of correctly classification rate which is 99.74%. This algorithm outperformed in the case of correctly classification rate. However, OlexGA shows the lowest correctly classification rate which is only 70.71%. For IBK, J48 and Naïve Bayes, they performed well by producing 99.16%, 98.91% and 99.07% correctly classification rate respectively. Compared with NB Tree + Random Forest and OlexGA, EGA has higher correctly classified values. A significant improvement was also found in correctly classification percentage. EGA shows 99.74% correctly classification which is higher than OlexGA as presented in Figure 5.12. EGA is also better compared to NB tree plus random forest.

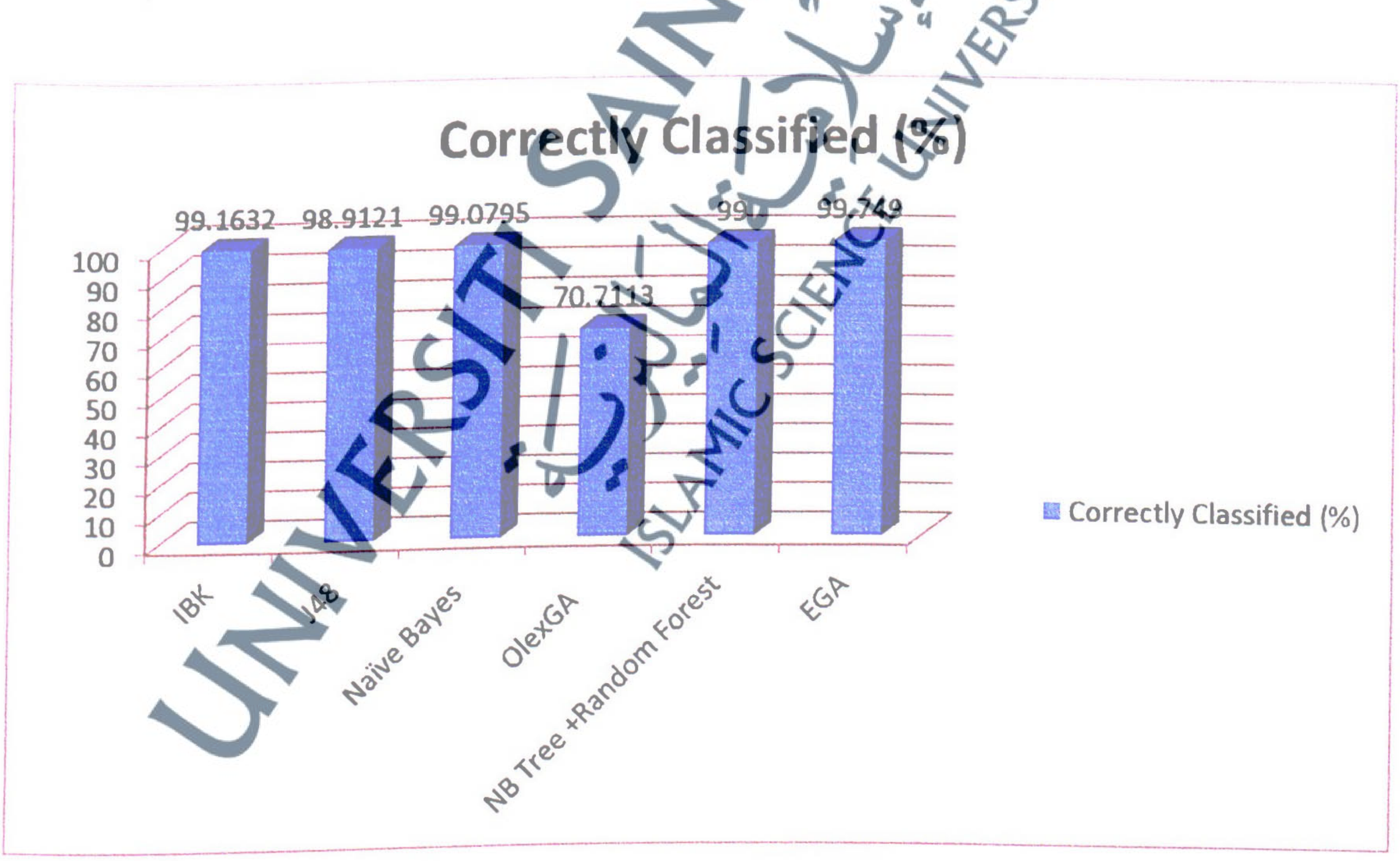

Figure 5.20: Correctly classified of various classification algorithms

Table 5.5 shows the overall experimental result collected from the output of data mining using Weka.

Table 5.5: Experimental results for different metrics of various algorithms.

Algorithm	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Correctly Classified (%)
IBK	0.992	0.006	0.992	0.992	0.992	1	99.1632
J48	0.989	0.016	0.989	0.989	0.989	0.998	98.9121
Naïve Bayes	0.991	0.03	0.991	0.991	0.991	0.981	99.0795
OlexGA	0.707	0.051	0.902	0.707	0.75	0.828	70.7113
NB Tree +Random Forest	0	1	0.99	0.991	0.99	0	99
EGA	0.997	0.014	0.997	0.997	0.997	0.992	99.749

Based on Figures 5.15 to 5.20 and the results presented in Table 5.5, it could be concluded that OlexGA does not perform better as compared to IBK, J48 and Naïve Bayer classifier. However, OlexGA is integrated with weka and tested with the proposed EGA technique, and this has improved this research one step ahead. The main goal of this research is to improve worm detection accuracy by using GA for cloud worm detection. To improve detection accuracy, it is necessary to focus on various parameters and used techniques in GA. This research employs various selection, crossover and mutation methods and parameters to find the best solution in order to improve worm detection accuracy in cloud.

GA works on natural selection process and the result shows that the EGA is 99.74% which is correctly classified. Accordingly, Figure 5.20 shows EGA algorithm classification rate with other established classification algorithms. As observed, EGA reached 99.74% accuracy rate which is higher compared to other algorithms.

From the results presented above, it is clearly seen that the proposed EGA performed better than OlexGA. OlexGA shows 70.71% accuracy rates where the proposed algorithm shows 99.74% accuracy rate. Hence, EGA shows significant improvement of 41.06% higher accuracy compared to OlexGA.

Based on the accuracy results, it shows that EGA is able to detect worm in cloud computing as compared to other algorithms. As seen in the table above, the value for TP has 0.997 which indicates high positive rate for EGA in cloud worm detection. As for FP value (0.014), it indicates low false positive rate for EGA in cloud worm detection. For the precision value (0.997), it indicates higher precision in relation to EGA in cloud worm detection. As for the recall value (0.997), it also indicates that EGA has higher recall value for cloud worm detection. As for the f-measure value (0.997), it shows that EGA has higher value in cloud worm detection. In conclusion, EGA has higher detection rate for cloud worm detection as compared to OlexGA and other algorithms.

5.8 Summary

From the experiment conducted using the same dataset, it was found out that OlexGA`s accuracy rate result is very low in cloud worm classification. Therefore, this research improves the performance of GA algorithm as classifier. GA is a search algorithm that is based on the natural selection principles and genetics. GA has the feature to generate new population from the existing population. By generating new population, it is possible to predict future attack type that can handle unknown threats in the future. A new genetic algorithm namely EGA was proposed in this chapter. To implement this algorithm the process goes through the security metrics, CIA, weight, and severity measurements. The proposed EGA algorithm shows 99.74% accuracy rate with 99.7% true positive rates and 1.4% false positive rate with 10 cross validation. The proposed EGA algorithm outperformed OlexGA.