Training a multi-criteria decision system and application to the detection of PHP webshells

Alexandre Croix
Royal Military Academy
Belgium
alexandre.croix@rma.ac.be

Thibault Debatty
Royal Military Academy
Belgium
thibault.debatty@rma.ac.be

Wim Mees
Royal Military Academy
Belgium
wim.mees@rma.ac.be

***Abstract*—In this paper we present an algorithm designed to train a multi-criteria decision system. This kind of system is very important and used a lot in different military fields and, particularly in cyber-defense. We developed this algorithm to be used with different multi-agent detection systems. The MASFAD system is a typical example [1]. It is a multi-agent system for Advanced Persistent Threat (APT) detection. In this paper we compare different optimization methods for learning Weighted Ordered Weighted Averaging (WOWA) coefficients in order to perform a binary classification. The WOWA function is an aggregation function that is a generalization of Ordered Weighted Averaging (OWA) and the Weighted mean. The WOWA operator combines both of their advantages The learning part is based on a Genetic Algorithm and uses a training dataset. We perform a complete parameter study and we determine the efficiency of our model by evaluating the performance during the classification of different PHP files as webshells or normal files. These PHP files were previously analyzed by a program developed at the Royal Military Academy. We obtain very accurate results and a good stability during the decision process. This system could be used in a lot of different fields.**

***Index Terms*—Webshell, machine learning, multi-criteria decision, aggregation functions**

I. Introduction

The growth of the amount of information and the usage of multi-criteria decision systems makes, data fusion techniques, and aggregation functions especially, more and more important. These techniques are used in a large field of applications and are related at two main problems: (i) the characterization of a model and (ii) the determination of parameters for a known function.

Several approaches have been developed to determine parameters for a known function. Some of them used a large dataset of examples, and parameters are deducted from these examples. An advantage of this kind of determination method is that is possible to resolve coefficients without the presence of domain experts to provide crucial information. Furthermore, data may come from different locations and, currently, for the majority of research fields, it is easy to obtain data.

In this paper we focus on a method to determine parameters for the WOWA function. WOWA, for Weighted Ordered Weighted Averaging, is an aggregation operator introduced by Vicen Torra in 1996 [2]. This operator is a generalization of Ordered Weighted Averaging (OWA) and the Weighted mean. Concretely, WOWA merges a set of numerical data in a single number thanks to two weighing vectors: one for the weighted mean (w) and the other for the OWA operator (p). The WOWA operator combines the advantages of both of them. The weighted mean, weights the information sources, and the OWA gives importance to the data according to their scores. In this work, we improve the quality of decision/classification. Another advantage is that the weights obtained for the WOWA function can be interpreted, which is not the case for deep-learning.

The WOWA function takes three vectors (w, p, $data$) in arguments to produce a single number. The expression could be represented by:

$$output = WOWA(w, p, data) \tag{1}$$

Rest of the paper is arranged as follow: Section II describes the structure of our algorithm with an explanation of the variants we developed: (i) two different population initialization methods, (ii) two evaluation criteria and (iii) two selection methods. In Section III we realize a performance evaluation of our algorithm by studying the impact of different parameters on the classification of PHP files as webshells or as normal files. Numerical data for PHP files classification are provided by a webshell-detector program, developed at the Royal Military Academy. In Section V we discuss results and give some leads to improve the work in the future. And, finally, we present our conclusion in Section IV.

II. Learning

Learning aggregation operator weights from training dataset is an optimization problem. In the particular case of weighted mean or OWA function, the problem is simply a quadratic minimization function. There are several different algorithms that solve the problem with accuracy. For the WOWA operator, the optimization problem is more complex. It exists some optimization techniques to approximate non-quadratic problems, but their implementation is difficult and non-trivial. In this case, [3], advises to use a Genetic Algorithm to learn the weights of the WOWA operator.

A Genetic Algorithm is an evolutive procedure that maintains a population of individuals $P(t) = (x_1^t, ..., x_n^t)$, for iteration t. Each element of the population, called a "chromosome", is a potential solution to the problem. A chromosome is composed of different characteristics, named "genes". Each

978-1-5386-9383-4/19/$31.00 ©2019 IEEE

chromosome is evaluated to measure its performance. The following generation, $t+1$, is generated by keeping the best chromosomes from the generation t and by reproducing them (crossover). Then, some chromosomes in the generation $t+1$ are randomly "mutated". The process is repeated until the algorithm reaches an end condition.

For our application, a gene is a single weight (value between 0 and 1) and a chromosome is an element composed of two weight vectors: w and p. Each vector contains several genes whose sum is mandatorily 1.

We now describe the different steps of our algorithm: (i) the population initialization, followed by a loop composed of (ii) an evaluation, (iii) a selection step, (iv) a reproduction step and (v) the mutation step.

A. Population initialization

There are several ways to initialize population. We implemented two, to measure their impact on the learning efficiency.

The initial population $P(0)$ is composed of N chromosomes, of two vectors, each, containing M genes, where M is the number of numerical data to aggregate (the number of data sources to merge).

1) Random initialization: The easiest method to perform an initialization population is simply to generate completely random chromosomes. Practically, we generate a random number (between 0 and 1) for each gene. Then, we normalize the two weight vectors independently to obtain chromosomes that respect the constraint. The following expressions are the equations used for the normalization:

$$w'_i = \frac{w_i}{\sum w_i} \tag{2}$$

$$p'_i = \frac{p_i}{\sum p_i} \tag{3}$$

2) Quasi-random initialization: We implement a "quasi-random" initialization method. The majority of chromosomes are randomly generated, but some of them are specific: all weights are equal to 0 except one that equals 1. For example, a specific chromosome with 2 weights in each vector $X = ((w_1, w_2), (p_1, p_2))$ could be $X = ((0, 1), (1, 0))$.

The goal of this initialization is to begin the algorithm with particular cases that could produce a better convergence to the solution.

B. Chromosome performance evaluation

For each generation, all chromosomes are individually evaluated in order to determine which ones give the best results. We implemented two different evaluation methods which have quite different functioning.

1) Distance performance evaluation: For each chromosome in the population, we compute the WOWA function on all the examples in the training dataset. Then, we compute the difference between the WOWA result just computed and the result given in the training dataset. All these differences are added to obtain the total distance of a chromosome that is the performance of this individual. The lower the distance is, better is the chromosome. To Obtain a distance of zero means the algorithm found a combination of weights that match perfectly with all examples in the training dataset.

2) Area Under the Curve performance evaluation: The Area Under the Curve (AUC) evaluation method is designed for binary classification problems and is directly related to the quality of detection. To obtain the AUC, we first build the receiver operating characteristic (ROC). The ROC is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It is created by plotting the *true positive rate* (true detection) against the *false positive rate* (false alarm) as shown in the Figure 1. The AUC, is the Area Under the ROC Curve. Better is the detection, greater is the Area Under the Curve (AUC).

In the case of the AUC evaluation performance criterion, for each chromosome in the population, we compute the WOWA function on all examples in the training dataset and, then, on these computed results, we perform the Area Under the Curve evaluation. This method gives directly an estimation of the classification efficiency for each chromosome. Unlike the previous method, higher is the result, better is the chromosome.

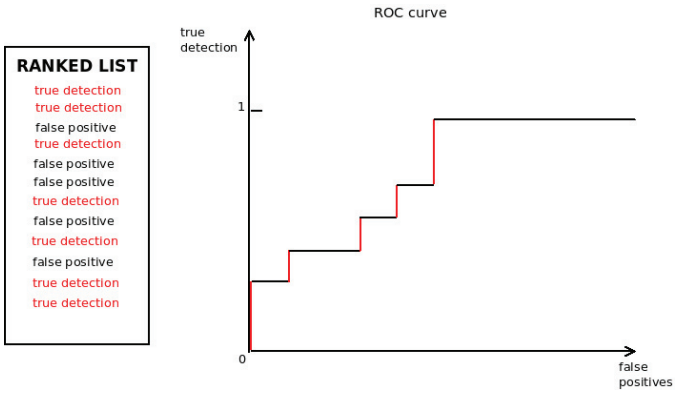

Fig. 1. Simple example of a ROC curve

The following pseudo-code describes how the AUC evaluation method works:

Algorithm 1 AUC chromosome performance evaluation

```
for all chromosomes in population do
    read w and p vectors from chromosome i
    auc ← 0
    wowa_output[]
    real_result[]
    for all examples in dataset do
        real_result[j] ← read example(j) from dataset
        wowa_output[j] ← wowa(w, p, example(j))
    end for
    auc ← computeAUC(wowa_output[], real_result[])
    chromosome.performance ← auc
end for
```

C. Selection step

In order to create a new generation $(t+1)$ of chromosomes, it is necessary to select some element from the generation t. The probability to select a chromosomes depends on its performance score evaluation. We implemented two methods to select "parents" for the next generation.

The number of chromosomes selected as parents depends on the crossover parameter. The crossover is the percentage of the population kept from the generation t to the generation $t+1$.

We also implemented an "elitism" principle: the two best chromosomes (compare to their performance evaluation score) from the generation t are systematically kept unchanged in the generation $t+1$. This "elitism" system prevents regressing from a generation to the next one. Indeed, in the worst case, the best result of a generation is equal to the best result of the previous generation.

We now describe different selection methods for Genetic Algorithm. In our work we used two of them : the Tournament Selection (TOS) and the Roulette Wheel Selection (RWS). During our tests, we did not note significant differences between the results with TOS or RWS.

1) Roulette Wheel Selection: This selection method gives to each individual i of the population a probability $p(i)$ of being selected proportional to its performance score $f(i)$. The expression of $p(i)$ is given by:

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \tag{4}$$

where n represents the number of elements in the population.

The process is repeated, without replacement, until the required number of parents is obtained.

According to [4], the Roulette Wheel Selection method could produce a premature convergence of the algorithm on a local minimum.

2) Tournament Selection: This method picks randomly two chromosomes and keeps, for the next generation, the chromosome with the best performance score. Chromosomes with good high performance score have more chance to be selected for the next generation. As the previous selection method, the process is repeated until obtain the necessary number of parents.

D. Reproduction step

During the reproduction step, chromosomes which have been selected during selection, are combined two-by-two to create new individuals. The reproduction procedure is detailed below and is inspired from [5].

First, a random number α is selected to determine the crossover point. This number is defined as:

$$\alpha = \lceil (rnd * M) \rceil \tag{5}$$

where M is the number of numerical data to aggregate and rnd is a random number between 0 and 1.

Two chromosomes are randomly selected:

$$P_{dad} = (p_{d1}, p_{d2}, ..., p_{d\alpha}, ..., p_{dM}) \tag{6}$$

$$P_{mom} = (p_{m1}, p_{m2}, ..., p_{m\alpha}, ..., p_{mM}) \tag{7}$$

The genes at the crossover point are combined together to create new genes:

$$p_{new1} = p_{m\alpha} - \beta[p_{m\alpha} - p_{d\alpha}] \tag{8}$$

$$p_{new2} = p_{d\alpha} + \beta[p_{m\alpha} - p_{d\alpha}] \tag{9}$$

where β is a random number between 0 and 1.

The final step is the combination of the two parents and the two new genes to obtain two new chromosomes:

$$child_1 = (p_{m1}, p_{m2}, ..., p_{new1}, ..., p_{dM}) \tag{10}$$

$$child_2 = (p_{d1}, p_{d2}, ..., p_{new2}, ..., p_{mM}) \tag{11}$$

The reproduction procedure is repeated to fill the population.

E. Mutation step

Mutation step is important to avoid converging too quickly on a local minimum [6]. Indeed, without mutation, generation after generation, the population will converge to a minimum. It is possible, even probable, that this minimum is a local minimum. Mutation produces "jump" on another position in the space of solutions and can discover new areas for the domain with better potential solutions.

Concretely, a random gene is selected and is replaced by a random value between 0 and 1. Then, the chromosome is normalized to respect the constraint (sum of weights equals 1).

III. Parameters study and experimental evaluation

A. Test setup

To evaluate the efficiency of our algorithm, we performed a parameters study[1] on data provided by a webshell detector. The detector analyses PHP files according to five different modules and gives a score, between 0 and 1 for each of them. The different modules are:

- **Signature** Checks if the file is known in a signature database. This kind of detection works well for known malicious file;
- **Fuzzy hashing**
- **Dangerous routines** Checks if the PHP file tries to execute some specific and potentially dangerous routines. For example: *exec*, *passthru*, *system*,...;
- **Obfuscation** Checks if the file executes some functions like *base64decode*, *rot13*,... A malicious file tries to hide the its content. The module measure also the longest string in the file;
- **Entropy**

[1]The complete source code and documentation are available at https://gitlab.cylab.be/cylab/wowa-training

The training dataset used for this work is composed of 12,468 PHP files that contains 206 PHP webshells. All these files were analyzed by the webshell-detector and the results were stored in serialized files.

The efficiency evaluation of a classification model needs two steps: (i) the learning part and (ii) the evaluation part. According to [7], it is very important to have different dataset for the learning and the evaluation. Generally, two-third of the data are used for the learning and a third for the evaluation.

In our case, it is not easy to find a sufficient amount of real different webshells. To overcome this problem we used a well-known data-mining method [8]: the *k-folds crossover validation*. This method consists of separing the dataset in k folds, performing the learning part on $k-1$ folds and then, evaluating the model on the last fold. The process is repeated k times by changing the fold used for the evaluation. All these k intermediate results are meant to obtain a general result. Taking k equals 5 or 10 usually produces, the best results.

Our training dataset has only 1.6% of malicious files. It is not possible to generate randomly the k folds. The probability to obtain very different repartitions would be too high. A fold could be composed of only one or two webshells.

Practically, we selected randomly, $\lfloor \frac{W}{k} \rfloor$ webshells, and $\lfloor \frac{F}{k} \rfloor$ "normal" files. Where W is the total number of webshells in the dataset, F is the total number of non-malicious files in the training dataset and k is the number of folds. We obtained k folds with similar proportion of webshells.

To raise the penalty to not detect a webshell file, we artificially increase the number of webshells in the learning dataset: concretely, in each learning dataset, we put 10 times each webshell scores.

To determine the efficiency of our algorithm, we performed a complete parameters study. We tested:

- **Population size** number of individuals in each population;
- **Crossover rate** percentage of the population kept to generate the next generation;
- **Mutation rate** percentage of genes which are mutated in each generation;
- **Generation number** number of generation before stopping the algorithm.

In combination to these four parameters, we have tested the two population initialization methods (random and quasi-random) and two performance evaluation criteria (distance evaluation and AUC evaluation).

To evaluate the performance of a parameters combination, we observed two measurements: (i) the average AUC obtained after the 10-folds cross validation and (ii) the range of threshold values in which the *true positive rate* and the *true negative rate* are better than 95%. In other words, it is the size range of threshold values that allows a correct classification for more than 95% of the files.

We measured this range in one specific fold: the fold, from the 10 folds of the cross-validation, with the best AUC value. Then, we compared this range with the range obtained by a classification generated by a mean as aggregation function.

B. Population number

For the first test, we varied the population number between 40 and 110 by step of 10. The other parameters were fixed and their values are:

Crossover rate	60
Mutation rate	15
Generation number	110

The Figure 2 presents the average AUC for all the values and for the four combinations of initialization and performance evaluation method.

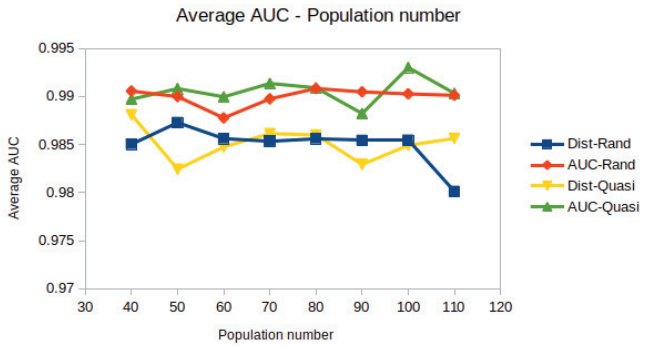

Fig. 2. Figure shows the variation of average AUC according to the variation of population number for the four combinations of initialization and performance evaluation method

The Figures 9, 10, 11 and 12 show the size range in which more than 95% of the files are correctly classified compared to the range size obtained with a classification performed thanks to a mean as aggregation. It is important to note that, because of the random generation of the folds, the results depend of the model but also of the composition of the folds. That can be observed on the Figure 17. We observe two curves with very similar behaviour but with different values. It means that some folds are "easier" to classify than others (similar behaviour) but the WOWA aggregation generally gives better results than a average aggregation.

C. Crossover rate

The second parameter studied is the crossover rate. The values varied from 10 to 90 by step of 10. The fixed values are:

Population size	100
Mutation rate	15
Generation number	110

As previously, Figure 3 shows the variation of the average AUC according to the crossover rate variation.

The Figures 13, 14, 15 and 16 compare the range size detection between the WOWA classification and a mean classification.

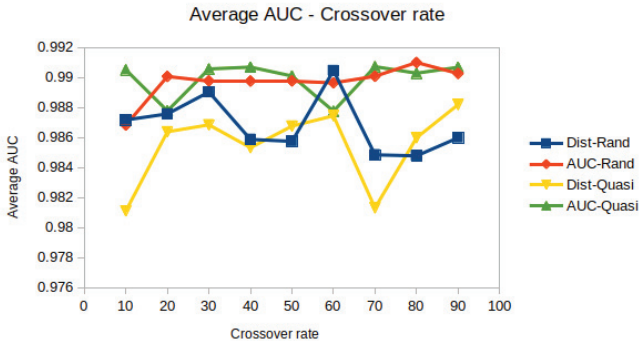

Fig. 3. Figure shows the variation of average AUC according to the variation of crossover rate for the four combinations of initialization and performance evaluation method

D. Mutation rate

The third parameter is the mutation rate. We tested the values between 5 and 50 by step of 5. The other fixed values are:

Population size	100
Crossover rate	60
Generation number	100

On Figure 4 we can see the impact of the mutation rate variation.

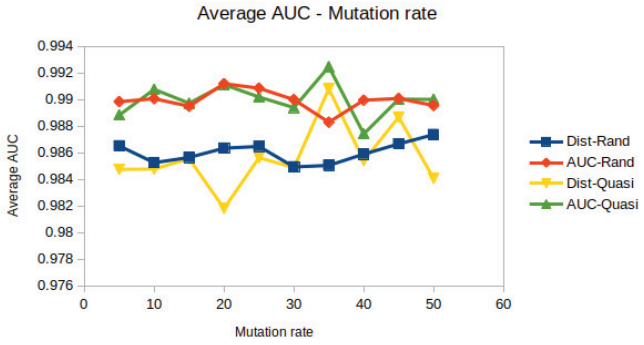

Fig. 4. Figure shows the variation of average AUC according to the variation of mutation rate for the four combinations of initialization and performance evaluation method

On the Figures 17, 18, 19 and 20 we can observe the impact of the mutation rate on the size of the detection range, compare to the range size for a classification that uses a simple average as aggregation.

E. Generation number

The way the algorithm is built, the best element of a generation is equal or better (according to the performance evaluation criterion) than the best element of the previous generation. Increasing the generation number can only improve the performance evaluation score. We use the learning curve to measure the evolution of the improvement. This curve represents the AUC value in function of the generation number. Note that for the AUC performance criterion, the learning curve is always increasing. That is not the case for the distance performance evaluation criterion. Indeed, that is not because the distance decreases that the AUC increases.

We generated the learning curves with fixed parameters:

Population size	100
Crossover rate	60
Mutation rate	15

The Figures 5 and 6 present the learning curve generated for the combination random initialization/AUC performance criterion and the quasi-random initialization/AUC performance criterion.

The two other combinations (random initialization/distance performance criterion and quasi-random initialization/distance performance criterion) are shown on the Figures 7 and 8. These figures also show the evolution of the distance criterion (that is always decreasing).

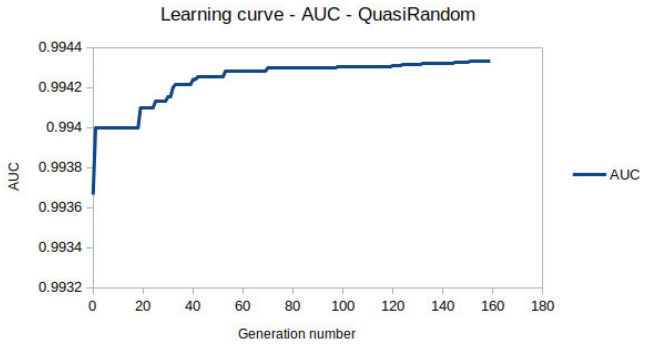

Fig. 5. Figure shows the learning curve for a model that uses a quasi-random initialization method and the AUC evaluation performance criterion.

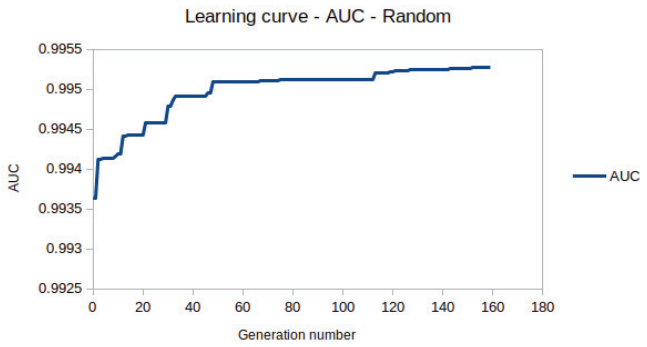

Fig. 6. Figure shows the learning curve for a model that uses a random initialization method and the AUC evaluation performance criterion.

We note that for all combinations (performance evaluation criterion/initialization method), running the algorithm longer than 100 generations does not really improve the results. The values are quite stable.

F. Synthesis results

This parameters study highlights the best values for the different parameters (whereas they are independent). We can note some important things:

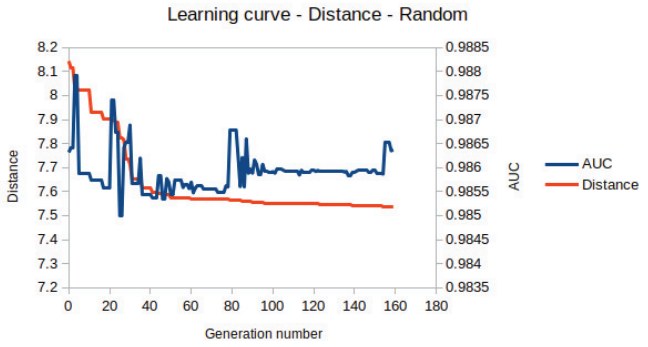

Fig. 7. Figure shows the learning curve for a model that uses a random initialization method and the distance evaluation performance criterion (blue) and the curve that represents the evolution of the distance value according to the generation number (orange).

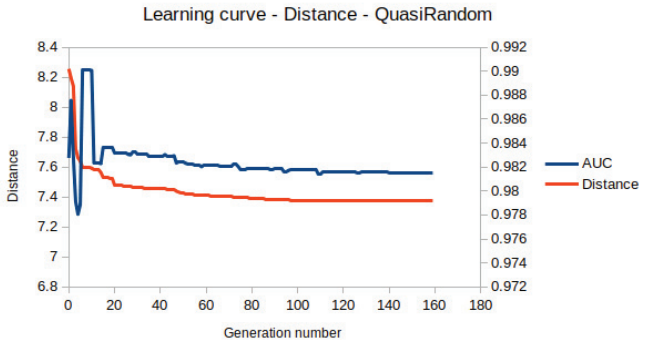

Fig. 8. Figure shows the learning curve for a model that uses a quasi-random initialization method and the distance evaluation performance criterion (blue) and the curve that represents the evolution of the distance value according to the generation number (orange).

- The best value of AUC is, obviously, obtained with the AUC performance evaluation criterion;
- The bigger sizes of detection range are obtained with the distance performance evaluation criterion. It means that the distance criterion is less sensitive than AUC criterion;
- The AUC criterion gives less good results, in term of range size, than the average classification;
- The repartition of weights depends a lot on the performance criterion used: with the distance criterion, the most important source (greatest weight) is the Signature module (between 45% and 90%). With the AUC criterion, the Signature has the smallest weight (around 3%);
- Quasi-random initialization method does not produce better results than random initialization;
- The population number parameter has no real influence on the results;
- The crossover rate parameter analysis shows the best value is 60%;
- The mutation rate parameter analysis shows the best value is 30%;
- The generation number parameter analysis shows that running more than 100 generations does not produce important improvement;

IV. Conclusion

Our work shows that it is possible to implement a multi-criteria based decision able to learn from a training dataset. We applied this algorithm on a realistic situation: classification of PHP files as a webshell or as a normal file. We obtained very interesting results: between 98% and 99% of the files are correctly classified. We obtained also a good improvement of the sensitivity of the classifier.

The algorithm also gives important information about the modules that are aggregated. It highlights the modules with a high importance and, mostly, the less important. This could point out that a module is inefficient and improve it in priority.

V. Future works

There are a lot of different ways to continue this work or to improve it. Currently, if we want to add some elements in the dataset examples, it is necessary to perform again all the learning part. It could be very interesting for the system to be able to adapt itself by modifying the weight values without a complete recomputation.

Our method to determine the best combination of parameters considers that all parameters are completely independent. It seems more realistic to suppose a correlation between the different parameters. This point could be improved in a future work.

The algorithm is built to be very generic. It will be easy to implement other aggregation functions or performance evaluation criteria.

As our algorithm can be used with numerical criteria, it could be interesting to adapt the system to work with non-numerical data.

References

[1] W. Mees and T. Debatty, "Multi-agent system for apt detection," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Nov 2014, pp. 401–406.

[2] V. Torra, "Weighted owa operators for synthesis of information," vol. 2, 10 1996, pp. 966 – 971 vol.2.

[3] D. Nettleton and V. Torra, "A comparison of active set method and genetic algorithm approaches for learning weighting vectors in some aggregation operators," *International Journal of Intelligent Systems*, vol. 16, pp. 1069–1083, 09 2001.

[4] K. Jebari, "Selection methods for genetic algorithms," *International Journal of Emerging Sciences*, vol. 3, pp. 333–344, 12 2013.

[5] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms with CD-ROM*. New York, NY, USA: Wiley-Interscience, 2004, pp. 56–60.

[6] ——, *Practical Genetic Algorithms with CD-ROM*. New York, NY, USA: Wiley-Interscience, 2004, p. 60.

[7] I. Witten and E. Frank, *Data Mining Practical Machine Learning Tools And Techniques*, 01 2005, vol. 11.

[8] T. Gunasegaran and Y. Cheah, "Evolutionary cross validation," in *2017 8th International Conference on Information Technology (ICIT)*, May 2017, pp. 89–95.

Fig. 9. Figure shows and compares the variation of the range size, according to the variation of the population number, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a random initialization and a distance evaluation criterion.

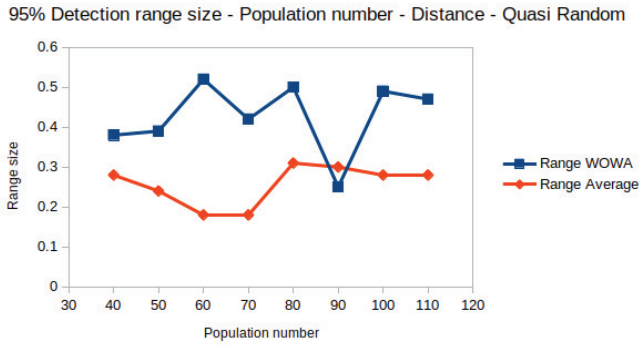

Fig. 10. Figure shows and compares the variation of the range size, according to the variation of the population number, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a quasi-random initialization and a distance evaluation criterion.

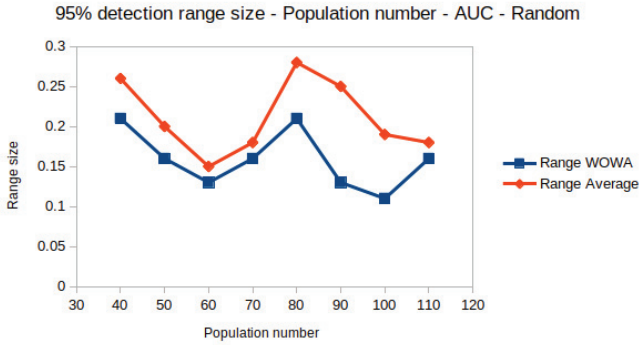

Fig. 11. Figure shows and compares the variation of the range size, according to the variation of the population number, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a random initialization and a AUC evaluation criterion.

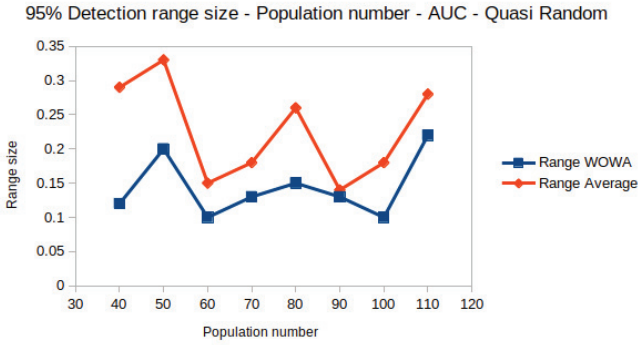

Fig. 12. Figure shows and compares the variation of the range size, according to the variation of the population number, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a quasi-random initialization and a AUC evaluation criterion.

Fig. 13. Figure shows and compares the variation of the range size, according to the variation of the crossover rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a random initialization and a distance evaluation criterion.

Fig. 14. Figure shows and compares the variation of the range size, according to the variation of the crossover rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a quasi-random initialization and a distance evaluation criterion.

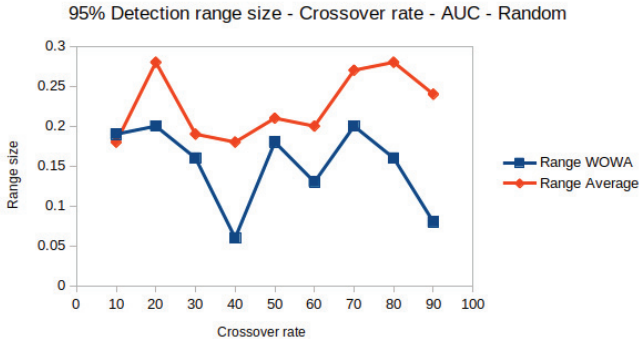

Fig. 15. Figure shows and compares the variation of the range size, according to the variation of the crossover rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a random initialization and a AUC evaluation criterion.

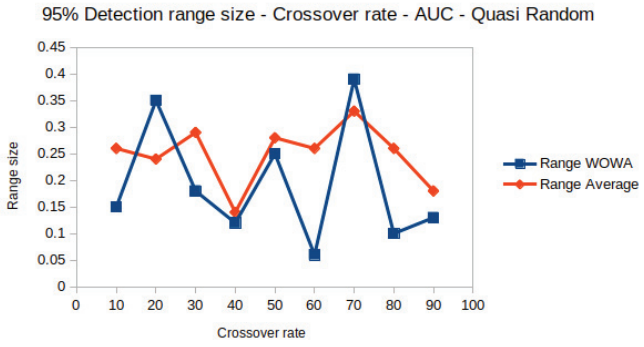

Fig. 16. Figure shows and compares the variation of the range size, according to the variation of the crossover rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a quasi-random initialization and a AUC evaluation criterion.

Fig. 17. Figure shows and compares the variation of the range size, according to the variation of the mutation rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a random initialization and a distance evaluation criterion.

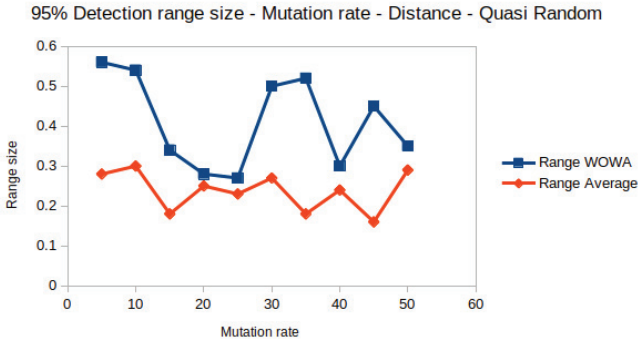

Fig. 18. Figure shows and compares the variation of the range size, according to the variation of the mutation rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a quasi-random initialization and a distance evaluation criterion.

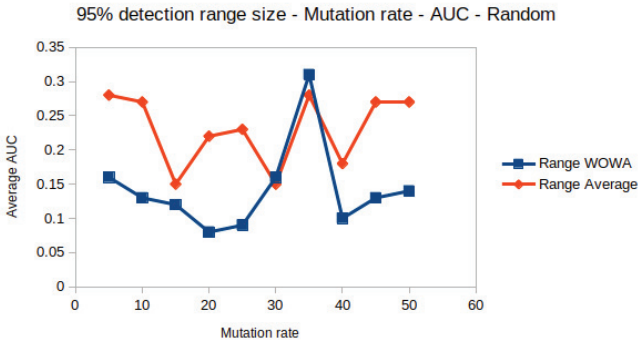

Fig. 19. Figure shows and compares the variation of the range size, according to the variation of the mutation rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a random initialization and a AUC evaluation criterion.

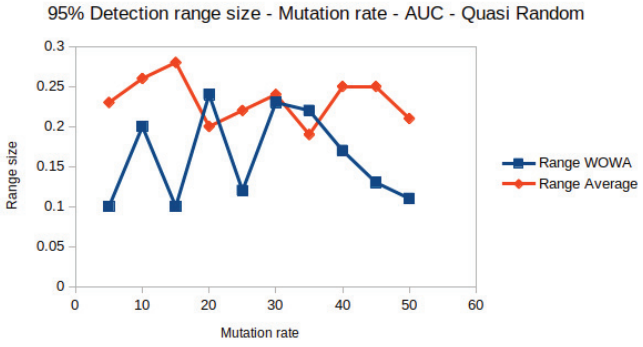

Fig. 20. Figure shows and compares the variation of the range size, according to the variation of the mutation rate, in which more than 95% of the files are correctly detected by using an WOWA aggregation function (blue) and an average aggregation (orange) with a quasi-random initialization and a AUC evaluation criterion.