Applying Cartesian Genetic Programming to Evolve Rules for Intrusion Detection System

Hasanen Alyasiri[1], John Clark[2] and Daniel Kudenko[1]
[1]*Department of Computer Science, University of York, U.K.*
[2]*Department of Computer Science, University of Sheffield, U.K.*

Keywords: Cartesian Genetic Programming, Intrusion Detection System, Stacking Ensemble.

Abstract: With cyber-attacks becoming a regular feature of daily business and attackers continuously evolving their techniques, we are witnessing ever more sophisticated and targeted threats. Various artificial intelligence (AI) algorithms have been deployed to analyse such incidents. Extracting knowledge allows the discovery of new attack methods, intrusion scenarios, and attackers' objectives and strategies, all of which can help distinguish subsequent attacks from legitimate behaviour. Amongst AI approaches, Evolutionary Computation (EC) techniques have seen significant application, particularly in the area of intrusion detection. In this paper, we show how one EC approach, namely Cartesian Genetic Programming (CGP), can construct rules (checks) for detecting malicious behaviour in a system. Experiments are conducted on up-to-date datasets and compared with state of the art approaches. We also introduce an ensemble learning paradigm, indicating how CGP can be used as stacking technique to improve learning performance.

1 INTRODUCTION

"Intrusion detection is the process of monitoring the events occurring in a computer system or network and analysing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices" (Scarfone and Mell, 2007). In recent years, security threats have had a major effect on the confidentiality, privacy and integrity of on-line services. Intrusion detection can help prevent or mitigate some of these threats. Intrusion detection relies extensively on the analysis of attack vectors and intentions. However, IDS systems face various problems such as large traffic volumes, unbalanced data distributions, the need to recognize normal and abnormal behaviour, and continuously changing environments (Wu and Banzhaf, 2010).

Researchers have applied a variety of machine learning and data mining approaches to address IDS problems. Evolutionary Computation (EC) techniques have been used by researchers to tackle numerous tasks in IDSs, for example searching for an optimal probe placement, automatic model design, and learning of classifiers (Wu and Banzhaf, 2010). EC approaches have a number of attractive features, such as producing readable outputs, producing lightweight rules, and an ability to make trade-offs between conflicting objectives (Sen, 2015). These features are very significant for security teams (Orfila et al., 2009). In addition, EC makes possible the automated synthesis of computer programs. It is often used when the desired solution is an 'expression' or program. EC-based techniques can be data distribution free, i.e., no prior knowledge is needed about the statistical distribution of the data and the approaches can operate directly on the data in their original form.

Various researchers have reported the use of some sort of EC for evolving IDS rules. In this paper we use a form of Genetic Programming (GP) called Cartesian Genetic Programming (CGP). CGP was introduced in (Miller and Thomson, 2000). CGP individuals take the form of directed acyclic graphs instead of the tree structures used by standard forms of GP. It represents these graphs as a two-dimensional grid of computational nodes (Miller, 2011). An individual's genotype is a linear string of integers that decodes to a directed graph. A set of predefined inputs and functions, encoded as graph nodes, are then sequentially evaluated ('executed') to give the desired output. CGP has been applied in a significant number of domains and problems (Miller and Smith, 2006). This paper demonstrates the use of CGP to discover intrusion detection rules.

Alyasiri, H., Clark, J. and Kudenko, D.
Applying Cartesian Genetic Programming to Evolve Rules for Intrusion Detection System.
DOI: 10.5220/0006925901760183
In *Proceedings of the 10th International Joint Conference on Computational Intelligence (IJCCI 2018)*, pages 176-183
ISBN: 978-989-758-327-8

Section 2 summarises past use of EC algorithms for IDS rule synthesis. Section 3 introduces our proposed method. Section 4 introduces the experimental datasets and the standard CGP implementation results. Section 5 introduces the stacking ensemble concept, implementation and results. The last section gives conclusions and identifies future work.

2 RELATED WORK

(Crosbie et al., 1995) gave the first GP application to intrusion detection, training autonomous agents to detect intrusive behaviours. Obvious intrusions that are misclassified during the evolution process are penalised heavily. (Folino et al., 2005) presented a GP ensemble for a distributed IDS. GP runs on a distributed hybrid multi-island model-based environment to monitor security-related activity within a network. Each island contains a cellular genetic program whose aim is to generate a decision-tree predictor, trained on the local data stored in the node and enhanced with the boosting algorithm AdaBoost.M2. Every genetic program operates cooperatively, yet independently of the others. After the classifiers compute their results, majority voting is used to form an ensemble. (Abraham et al., 2007) use three GP variants: linear genetic programming, multi-expression programming and gene expression programming to build an intrusion detection program. The evolved detectors are effective against a variety of attacks such as denial of service, probe, user-to-root and remote-to-local attacks. The work demonstrated the ability of GP techniques to develop lightweight and accurate detection rules compared to some of the conventional intrusion detection systems based on machine learning paradigms. (Sen and Clark, 2011) applied GP and Grammatical Evolution (GE), another evolutionary computation paradigm, to build IDS rules for deployment in mobile ad hoc networks (MANETs). Their research shows that GP and GE can be used to evolve efficient detectors automatically for known attacks, namely ad hoc flooding and route disruption against the routing protocol on MANETs. The experimental outcomes show that GP and GE are good at discovering complex relations on MANET data and that GP gives a better performance than GE under their (approximation to) optimal parameter settings.

3 CARTESIAN GENETIC PROGRAMMING

"CGP is a form of automatic evolution of computer programs and other computational structures using ideas inspired by Darwin's theory of evolution by natural selection" (Miller, 2011). In CGP, a program is encoded as a linear string of integers representing a directed graph. Each program is divided into subsets of genes (i.e. genotype) of the same length, representing the nodes of the graph.

The genotype describes from where a node receives its data, what operations the node carries out on the data, and how user output data is extracted. When the genotype is decoded, some nodes are ignored. This happens when a node's outputs are not subsequently used in the calculation of output data. When this happens, we refer to the nodes and their genes as 'inactive'. Previous investigations (Miller and Thomson, 2000)(Miller and Smith, 2006) have shown how a significant percentage of inactive nodes can help the efficiency of the evolution process. Unlike a tree structure, where there is always a unique path between any pair of nodes, graphs permit more than one path between any pair of nodes. If we assume all nodes perform some computational function, graph representations of functions are more compact than tree representations since they permit the reuse of previously calculated subgraphs. EC algorithms are also susceptible to bloat, a phenomenon where a large portion of the evolved program code has no influence on the fitness but whose execution still consumes resources (and so typically extends execution times). CGP avoids such bloat (Miller and Smith, 2006).

CGP's genotype decoding process is recursive in nature. Decoding starts with the output genes. CGP programs may have as many output nodes as necessary. These final output nodes are deemed to be 'active'. The decoding process identifies the nodes whose outputs are used as inputs to these nodes. These input nodes themselves become 'active' and the same procedure is repeatedly applied until the full function is identified, ending with the identification of appropriate terminal input nodes. The decoding process extracts the active nodes; inactive nodes are not processed and so having inactive genes presents little computational overhead. Consider the example shown in Figure 1.

There are three input nodes, which are indexed by 0,1, and 2. (These do not form part of the genotype, but may be indexed by it.) The remaining (computational) nodes of the system are now numbered contiguously, from 3 to 8. System inputs and computational

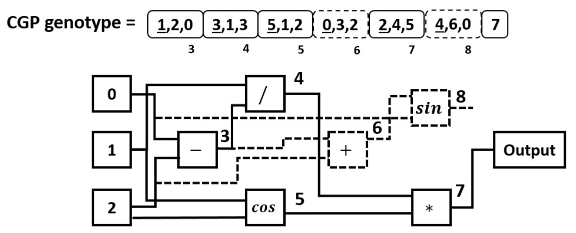


Figure 1: CGP genotype and corresponding phenotype.

nodes are therefore numbered contiguously over the range 0 to 8.

The genotype contains structures of three integers for each of its computational nodes. The underlined genes in the genotype encode the specific function for each node. There are six possible functions, denoted by the integers 0 to 5: add (0), subtract (1), multiply (2), divide (3), sin (4) and cos (5). The remaining integers in each node structure are associated with terminal inputs or with other function nodes that are located to the left. The number of actual inputs depends on the arity of the function. The number of formal inputs to a computational node is the maximum arity of any function in the function set. Any extra inputs will be neglected by function nodes that require fewer inputs than this maximum.

The last nodes in the genotype identify the output nodes. Here there is only one such node, which takes the value 7. The choice of node 7 as the output node coupled with the value of earlier node triples induces the active and inactive status of all earlier nodes.

Crossover operators have not been widely adopted in CGP. Originally, a one-point crossover operator was used (similar to the n-point crossover in genetic algorithms) but was found to disrupt the subgraphs within the chromosome (Miller, 2011). In another investigation (Clegg et al., 2007), a new floating-point crossover operator was found to improve performance for symbolic regression problems. However, further work is needed on a range of problems in order to evaluate its advantages. In a point mutation, a value at a randomly chosen gene location is changed to another valid random value. If a function gene is selected for mutation, then a valid value is the label of any function in the function set, whereas if an input gene is selected for mutation, then a valid value is the label of the output of any previous node in the genotype or of any program input. In addition, a valid value for a program output gene is the label of the output of any node in the genotype or the label of a program input. The user defines the number of genes in the genotype that can be mutated in a single application of the mutation operator. An example of the point mutation operator is shown in figure 2. A single point mutation takes place in the program gene, altering the output node input connection from 7 to 8. This makes nodes 6 and 8 to become active, whilst making nodes 4, 5 and 7 inactive. The inactive areas are shown in dashes. This highlights how a small change in the genotype can sometimes produce a large change in the phenotype.

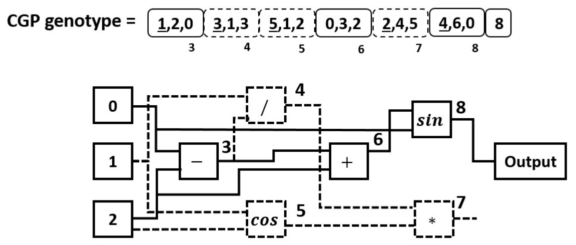


Figure 2: CGP genotype and corresponding phenotype after mutation.

For individual selection the (1+ λ) Evolutionary Strategy (ES) is normally used in CGP. Usually λ is chosen to be 4. ES configures a single parent to breed four children using mutation at each generation. The best individual among the parent and children is kept in the next generation and the process is repeated. In an ES approach, child genotypes are favoured over the parent. The parent is replaced by one of its offspring when offspring genotypes have the same fitness as the parent and there is no offspring that is better than the parent. This is an important characteristic of the algorithm, which makes good use of redundancy in CGP genotypes (Miller, 2011).

The method we propose uses a supervised learning CGP algorithm for detecting the Internet and computer network threats. We will use existing datasets for which reasonable features have already been extracted. The training phase will finish when one of two criteria met: either all instances are categorised correctly; or an identified maximum number of generations have been produced. Then, the best evolved rules are evaluated in a test phase.

3.1 Parameter Settings for CGP

The ECJ (Luke, 1998) toolkit is used for the CGP implementation. The population size chosen was 5. CGP generally uses very small population sizes and large numbers of generations (Miller, 2011). The number of generation was 10,000, except for the modern DDoS dataset where a value of 2,000 was adopted. This is because we did not witness any change in the fitness value after reaching 2,000 generation for all runs. The mutation rate used was 0.01. The maximum number of nodes was 500. (This does not include inputs and outputs nodes.) These settings were determined empirically via preliminary experimentation. We do not claim that these values are optimal.

Function nodes consist of a set of mathematical, relational and logical operators. The single input arity functions used were *sin*, *cos*, *log*, *log*1*p*, *sqrt*, *abs*, *exp*, *ceil*, *floor*, *tan* and *tanh*. The binary functions were $+$, $-$, $*$, protected $(/)$, *power*, *max*, *min*, *percent*, $>$, $\geq$, $\leq$, $<$, $=$, $\neq$, AND and OR. The rest of the parameters were determined automatically by the packages. In our implementation, we evolve two output nodes and compare them using the " $>$ " function to obtain a binary classification decision (normal vs. anomaly).

3.2 CGP Performance Measure

The effectiveness of proposed algorithms can be measured according to how malicious and normal behaviours are classified. Our datasets are labelled as normal or anomalous. We calculate the numbers of: true positives (TP), where malicious events are accurately classified as such; true negatives (TN), where normal events are classified as such; false positives (FP), where normal events are classified as malicious; and false negatives FN), where malicious events are classified as normal. Both false classifications are problematic. FPs waste a great deal of time and can lead to loss of confidence and FN are examples of the detector system not performing its primary task, i.e. they are attacks that go undetected.

Three derived performance metrics are used to assess our IDS optimize rule. The Detection Rate (DR) $(\frac{TP}{(TP+FN)})$ indicates the fraction of real attacks that are detected; this is sometimes is referred to as Recall. Accuracy $(\frac{(TP+TN)}{(TN+TP+FN+FP)})$ defines the fraction of all instances (attacks or non-attacks) that are correctly classified. IDS experts consider both False Positive Rate (FPR) $(\frac{FP}{(FP+TN)})$ and False Negative Rate (FNR) $(\frac{FN}{(FN+TP)})$ to be as important as the detection accuracy. Finally, the False Alarm Rate (FAR) $(\frac{(FPR+FNR)}{2})$ gives the rate of misclassified instances.

3.3 Cost Function

The cost function measures how much an individual's performance deviates from the ideal. The cost function used in our experiments is defined as $((\frac{TP-Anomaly\ Count}{Anomaly\ Count})^2 + (\frac{TN-Normal\ Count}{Normal\ Count})^2)$. When classification is perfect the cost is 0.

4 EXPERIMENT

The performance of the proposed techniques have been evaluated using four different datasets: Kyoto 2006+ (Song et al., 2011), Phishing websites (Mohammad et al., 2015), UNSW-NB15 (Moustafa and Slay, 2015), and modern DDoS (Alkasassbeh et al., 2016). These datasets are fully labelled and contain realistic normal and malicious scenarios. Extracted features are numeric, symbolic or binary. However, we did not employ all sets of features available for different aspects. We did not want to increase learning algorithm complexity by processing symbolic features such as protocol type, packet type and flag. Some attributes, such as IP addresses and port numbers, may be changing constantly for a variety of reasons. Any detectors relying on these attributes may not generalise well in real world applications. To obtain a statistically significant measure that does not depend on the initial random seed, CGP algorithm results correspond to an average of 20 independent runs. The results from the testing phase are presented here.

4.1 Description of the Benchmark Datasets

The datasets used in this study are publicly accessible labelled and suitable datasets for training and testing intrusion detection system. Four widely different types of environment are used to generate them. The scientific community is currently using these datasets. More details on the feature names, types, possible values and descriptions of environments and threats are given in the corresponding references.

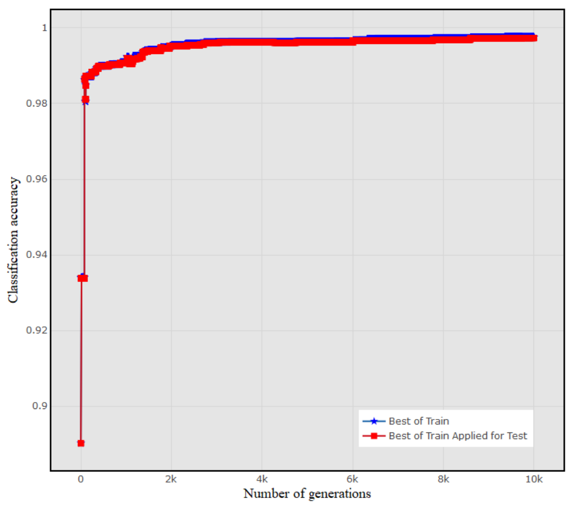
To allow a fair comparison with the published results of other detection systems, the same dataset samples and splitting percentages (training/testing) are chosen for evaluation . The relation between accuracy of the best evolved program out of all runs from each dataset and number of generations is examined and demonstrated in figure 3. The best program shows a performance on the testing dataset almost as good as that on the training dataset, except on UNSW-NB15 dataset due to its complexity (Moustafa and Slay, 2016).

4.2 Results

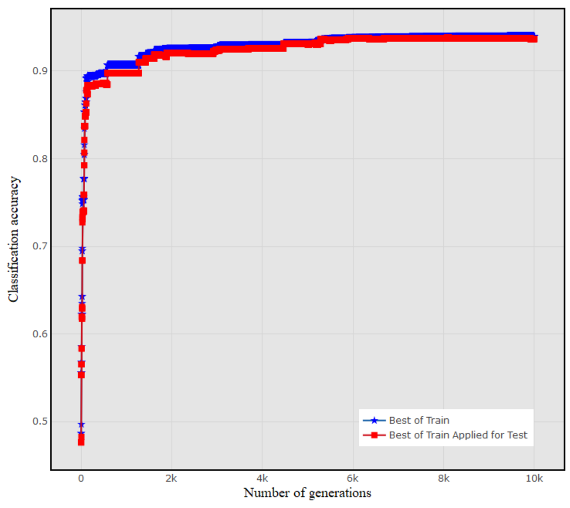
Table 1 shows the performance of the evolved rules.

Table 1: Proposed method performance (%).

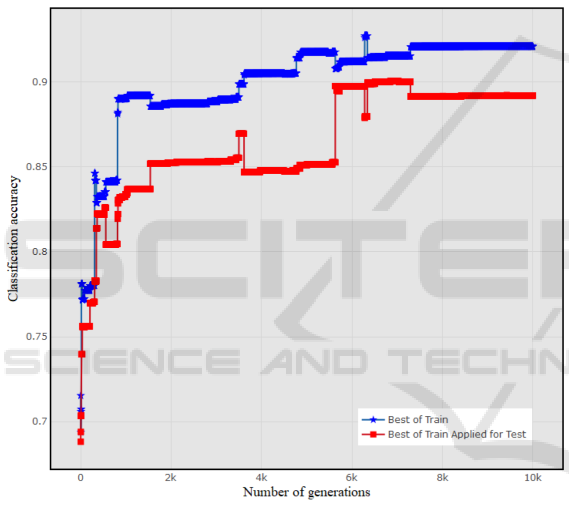
Dataset	**CGP**		
	DR	**FAR**	**Accuracy**
Kyoto 2006+	99.65	0.70	99.35
Phishing Websites	89.99	8.31	91.88
UNSW-NB15	94.20	13.46	87.31
Modern DDoS	87.38	7.13	97.19



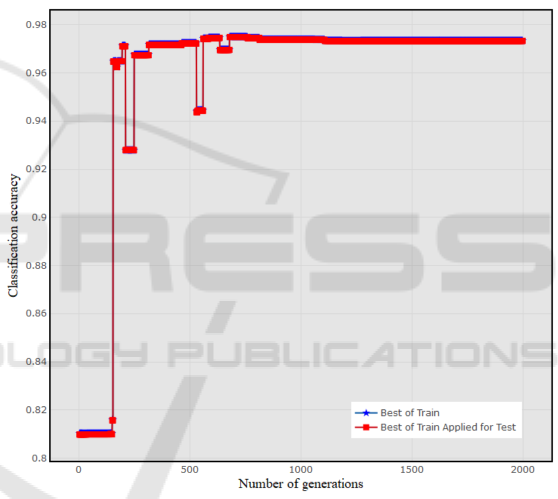
(a) Kyoto 2006+



(b) Phishing Websites



(c) UNSW-NB15



(d) Modern DDoS

Figure 3: CGP: Relation between classification accuracy and number of generations from each of the datasets.

To demonstrate the performance of the proposed method, results obtained by CGP are compared with those of other paradigms. (Ambusaidi et al., 2016) build an IDS using Least Square Support Vector Machine algorithm integrated with feature selection algorithm Flexible Mutual Information Based Feature Selection (LSSVM-IDS+FMIFS). The main idea is to remove redundant and irrelevant features in data, as these features not only slow the classification process but also affect IDS classification accuracy. The evaluation showed that feature selection algorithm contributes more critical features for LSSVM-IDS to achieve better accuracy and lower computational cost. The performance of LSSVM-IDS+FMIFS tested on Kyoto 2006+ dataset were 99.64% DR, 0.13% FPR ,and 99.77% accuracy, while CGP accomplished 99.65% DR, 1.06% FPR, and 99.35% accuracy. To combat phishing website threats, (Thabtah et al., 2016) proposed anti-phishing model deploying improved Self-Structuring Neural Network (iSSNN) combined with a feature selection method. Information Gain, Chi-Square and Gain Ratio have been utilized for the purpose of feature selection. iSSNN when epoch size is set to 500 combined with Information Gain for preprocessing achieved the higher performance in most cases. iSSNN-500 experiment results were 93.06%, 92.30%, 91.12%, 93.71% of accuracy, F1-score, recall and precision respectively. This compare to 91.88%, 90.69%, 89.99%, and 91.44% achieved by CGP. (Moustafa and Slay, 2016) have implemented Network Anomaly Detection System (NIDS) using five techniques, namely, Naïve Bayes, Decision Tree (DT), Artificial Neural Network, Logistic Regression, and Expectation-Maximisation Clustering. NIDS was tested on nine types of modern attacks fashions and new patterns of normal traffic. The DT technique accomplishes the highest accuracy (i.e. 85.56%) and the lowest FAR (i.e. 15.78%) tested on UNSW-NB15 dataset. Finally, (Alkasassbeh et al., 2016) proposed an IDS to detect and classify Distributed Denial of Service Attacks (DDoS) using three well-known data mining techniques: Multilayer Perceptron (MLP), Naïve Bayes and Random Forest. The IDS were evaluated on the modern DDoS dataset. The experimental results show that MLP achieved the highest accuracy of 98.63%.

So far, we have focused on developing an IDS using standard algorithm implementations. However, modern IDS model building can be integrated with a wide variety of machine learning methods. For instance, feature selection, various ensemble and hybrid techniques, and these methods generally produce favourable results (Aburomman and Reaz, 2017). These approaches are commonly used combined with a single stage learner. An ensemble approach generally produces better performance than the individual classifiers. Another advantage of ensemble approaches is their design. Ensembles allow substitution one or more algorithms with more accurate ones. Consequently, we adopt an ensemble technique in our learning phase.

5 ENSEMBLE

The concept of combining a collection of weak learners and forming a single, strong learner is referred to as an ensemble. An ensemble may combine multiple (heterogeneous or homogeneous) models to obtain reliable and more accurate prediction. Different schemas can be considered to generate the detectors and to combine them, i.e. the same learning algorithm can be trained on different datasets or/and different algorithms can be trained on the same dataset (Folino and Pisani, 2015). Popular conventional ensemble methods include bagging, boosting and stacking (Aburomman and Reaz, 2017). Stacking is the abbreviated term used to refer to Stacked Generalization (Wolpert, 1992). The main idea behind stacking is finding the optimal combination of a collection of base learners. Stacking is a class of algorithms that involves training a second level "meta-learner" to find the combination. Unlike bagging and boosting, the goal in stacking is to combine strong, diverse sets of learners. Moreover, ensemble techniques like bagging and boosting are often used to generate homogeneous ensembles, whereas stacking can be used to produce heterogeneous ensembles.

The algorithms used to generate the pool of classifiers for this experiment are implemented in the H2O.ai platform (H2O.ai, 2018). H2O.ai includes many common machine learning algorithms and implements best in class algorithms at scale. It also includes a stacked ensemble method that finds the optimal combination of a collection of prediction algorithms using stacking. The Automatic Machine Learning (*AutoML*) function in the platform is used to build base learning models. This function automates the supervised machine learning model-training process. AutoML performs basic data preprocessing if required such as imputation, one-shot encoding and standardization. During the model generation stage, it implements random grid search and model parameter tuning using the validation set to produce best performing models. AutoML trains and cross-validates a Distributed Random Forest (DRF), an Extremely-Randomized Forest (XRT), a random grid of Gradient Boosting Machines (GBMs), a fixed grid of Generalized Linear Model (GLM), a random grid of Deep Neural Nets (DNNs). In AutoML, the maximum number of models to generate is set to 10 and the rest of the parameters determined automatically by the platform. Note it is possible to produce more than one model from the same family. However these models may have different parameters.

5.1 Evolving Stacking through CGP

In this work, various algorithms are applied to the same dataset to create different learner models. After a number of models are created usually utilizing part of the dataset i.e. training, the decisions of these models are combined and a common decision is taken. In phase two, CGP is applied to create the actual ensemble model. Ensembles are coded as genetic programs, each individual act as a possible aggregation of the available base learners. More specifically, each ensemble is represented as a graph, while input vector elements (i.e. base learners) are input nodes. We considered the continuous outputs of the base learners, rather than the discrete outputs. This ensemble model is then used to predict class labels for a new dataset (i.e.the testing dataset). To avoid a further phase of training, we split the training dataset into 80% training used to train base learners and 20% validation set used to build stacked ensemble model through CGP (see figure 4).

CGP offers an intelligent self-configuration environment to perform dynamic model selection and integration for constructing the ensemble. The

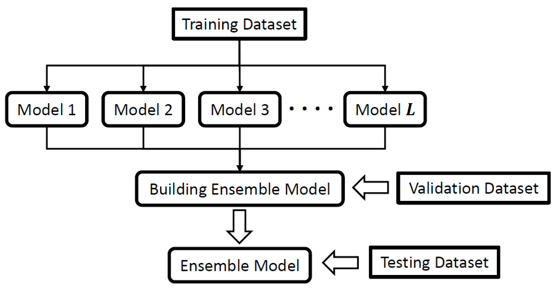


Figure 4: Proposed ensemble.

CGP parameter settings introduced in section 3 are extended to evolve a detection rule of the ensemble model. However, all CGP stacking runs used 2,000 generations. An example of CGP generated output of the stacking model (taken from UNSW-NB15 dataset) are shown in figure 5. Besides being readable output and ready for deployment, the solution showed the 6 learner models that were selected out of 10. This helps experts to understand how the ensemble model works. Furthermore, it can be seen from this figure that the *cos GBM_Model_4* node is reused 3 times in this rule which shows one of CGP graph representation's advantages.

$$o0 = sqrt\ (-\ (cos\ GBM\_Model\_4)\ GBM\_Model\_3)$$
$$o1 = log\ (/\ (power\ (+\ (sqrt\ DNN)\ GBM\_Model\_1)\ (cos\ GBM\_Model\_4))\ (sqrt\ (exp\ (min(exp\ XRT)\ (min\ (sqrt\ (cos\ GBM\_Model\_4))(=\ GBM\_Model\_3\ DRF))))))$$

Figure 5: CGP stacking output.

5.2 Stacking Experiment Results

In practice, the default cross-validation folds in AutoML is 5 while training the base learners. The results of applying the proposed technique to the testing datasets are shown in table 2. Comparing the standard and ensemble implementations, it can be seen that adopting the ensemble learning phase helped to enhance the DR, reduce FAR and improve overall accuracy. The most striking result to emerge was the CGP performance on the phishing dataset. In this dataset, our CGP stacking can provide a score of 96.29% F1-score and 95.28% precision. CGP demonstrated a superior performance for all datasets except for modern DDoS. Although CGP stacking managed to improve DR compared to standard CGP, FAR and accuracy performances slightly declined. It is worth noting that this is the only unbalanced dataset distribution and we treated both classes as equally important in this implementation. Finally, ensemble synthesis by means of stacking heterogeneous base learners produced the best classifier performance compared with other detection systems tested on the same dataset in most cases.

Table 2: Stacking method performance (%).

Dataset	**CGP Stacking**		
	DR	**FAR**	**Accuracy**
Kyoto 2006+	99.94	0.14	99.87
Phishing Websites	97.33	3.22	96.70
UNSW-NB15	96.29	11.29	89.47
Modern DDoS	87.40	7.18	97.10

6 CONCLUSIONS

This paper investigates an application of CGP to the automated discovery of intrusion detection rules. CGP has not been previously considered for this problem. We used four publicly available datasets. These datasets contain contemporary threats gathered from various environments. Our proposed approaches are capable of mapping the input vector space into a decision space to allow effective discrimination between classes (i.e. normal vs. anomaly). In addition, we have considered the use of stacking to ensemble synthesis to improve intrusion detection performance. CGP stacking was generally able to generate, given a domain, a good stacking configuration. Our approach determines not only which base learners must be present but also the combining method of these learners. The obtained results highlight clearly the benefits of adopting stacking heterogeneous ensembles over other paradigms.

One interesting feature of the approach is the ability to evolve a detector that can detect a wide range of threats. However, we now intend to investigate CGP ability to cope with new attacks (i.e. zero-day attacks). Though we have not done so here, it should also be possible to examine how the use of complementary classifiers can be applied to the problem at hand (i.e. we would aim to synthesise classifiers for complementary weaknesses).

ACKNOWLEDGEMENTS

Hasanan Alyasiri would like to thank the Iraqi Ministry of Higher Education and Scientific Research and the University of Kufa for supporting his PhD study.

REFERENCES

Abraham, A., Grosan, C., and Martin-Vide, C. (2007). Evolutionary design of intrusion detection programs. *IJ Network Security*, 4(3):328–339.

Aburomman, A. A. and Reaz, M. B. I. (2017). A survey of intrusion detection systems based on ensemble and hybrid classifiers. *Computers & Security*, 65:135–152.

Alkasassbeh, M., Al-Naymat, G., Hassanat, A. B., and Almseidin, M. (2016). Detecting distributed denial of service attacks using data mining techniques. *International Journal of Advanced Computer Science and Applications*, 7(1).

Ambusaidi, M. A., He, X., Nanda, P., and Tan, Z. (2016). Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE transactions on computers*, 65(10):2986–2998.

Clegg, J., Walker, J. A., and Miller, J. F. (2007). A new crossover technique for cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1580–1587. ACM.

Crosbie, M., Spafford, G., et al. (1995). Applying genetic programming to intrusion detection. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 1–8. Cambridge, MA: MIT Press.

Folino, G. and Pisani, F. S. (2015). Combining ensemble of classifiers by using genetic programming for cyber security applications. In *European Conference on the Applications of Evolutionary Computation*, pages 54–66. Springer.

Folino, G., Pizzuti, C., and Spezzano, G. (2005). Gp ensemble for distributed intrusion detection systems. *Pattern Recognition and Data Mining*, pages 54–62.

H2O.ai (2018). *h2o: R Interface for H2O*. Package version 3.18.0.2.

Luke, S. (1998). ECJ evolutionary computation library. Available for free at http://cs.gmu.edu/∼eclab/projects/ecj/.

Miller, J. F. (2011). Cartesian genetic programming. In *Cartesian Genetic Programming*, pages 17–34. Springer.

Miller, J. F. and Smith, S. L. (2006). Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *European Conference on Genetic Programming*, pages 121–132. Springer.

Mohammad, R. M., McCluskey, L., and Thabtah, F. (2015). UCI machine learning repository: Phishing websites data set. https://archive.ics.uci.edu/ml/datasets/Phishing+Websites.

Moustafa, N. and Slay, J. (2015). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015*, pages 1–6. IEEE.

Moustafa, N. and Slay, J. (2016). The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Information Security Journal: A Global Perspective*, 25(1-3):18–31.

Orfila, A., Estevez-Tapiador, J. M., and Ribagorda, A. (2009). Evolving high-speed, easy-to-understand network intrusion detection rules with genetic programming. In *Workshops on Applications of Evolutionary Computation*, pages 93–98. Springer.

Scarfone, K. and Mell, P. (2007). Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94.

Sen, S. (2015). A survey of intrusion detection systems using evolutionary computation. *Bio-Inspired Computation in Telecommunications*, pages 73–94.

Sen, S. and Clark, J. A. (2011). Evolutionary computation techniques for intrusion detection in mobile ad hoc networks. *Computer Networks*, 55(15):3441–3457.

Song, J., Takakura, H., Okabe, Y., Eto, M., Inoue, D., and Nakao, K. (2011). Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 29–36. ACM.

Thabtah, F., Mohammad, R. M., and McCluskey, L. (2016). A dynamic self-structuring neural network model to combat phishing. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 4221–4226. IEEE.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259.

Wu, S. X. and Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1–35.