The **A**rtificial **L**ife and **A**daptive **R**obotics Laboratory
ALAR Technical Report Series

Can Evolutionary Computation Handle Large Dataset?

Hai H. Dam, Kamran Shafi, Hussein A. Abbass

TR-ALAR-200507011

The Artificial Life and Adaptive Robotics Laboratory
School of Information Technology and Electrical Engineering
University of New South Wales
Northcott Drive, Campbell, Canberra, ACT 2600
Australia
Tel: +62 2 6268 8158 Fax:+61 2 6268 8581

Can Evolutionary Computation Handle Large Dataset?

Hai H. Dam, Kamran Shafi, Hussein A. Abbass
School of Info. Tech. and Electr. Eng.
Univ. College, Univ. of New South Wales
ADFA, Canberra ACT 2600
Australia
Email: {z3140959, k.shafi, abbass}@itee.adfa.edu.au

Abstract

Evolutionary Learning Classifier Systems (ELCS) were introduced by Holland a few decades ago. Since their birth, they were successfully applied to various data analysis domains. XCS is currently considered as state of the art ELCS. Earlier work have experimented with XCS on artificial problems or small datasets, and shown good results. However, XCS has not been tested on large datasets, particularly in the intrusion detection domain. This work investigates the performance of XCS on the 1999 KDD Cup intrusion detection dataset, a real world dataset approximately five million records, more than 40 fields and multiple classes with non-uniform distribution. Intuitively XCS fits well into this domain due to its online learning mode and its ability to evolve accurate and maximum general rules. We report the findings on experiments with the original XCS that show low accuracy. We then propose several modifications to XCS to improve its detection accuracy. The modified XCS improves remarkably on three of the four classes. The overall accuracy is equivalent to that of traditional machine learning algorithms, with the additional advantages of being evolutionary and online learner.

I. Introduction

XCS was introduced by Wilson [17], [18] as an enhanced version of the traditional ELCS proposed by Holland [9]. It is widely accepted as one of the most reliable Michigan-style ELCS for machine learning [2], [5]. The two major changes of XCS are: a fitness based on the accuracy of the reward prediction instead of the strength (or reward directly received from the environment) using a reinforcement learning approach; and a niche Genetic Algorithm (GA) to improve the spread of the state-action table. Many studies showed that XCS performs at least as well as other traditional machine learning techniques on several small data mining problems [1], [15]. The advantages of XCS can be counted as being a rule-based representation, which makes it easier to recognize the mined patterns; and being an online-learner with linear O(n) complexity. In other words, XCS sees each training instance only once unlike other learning mechanisms such as C4.5 or neural network classifiers that learn in batch mode and thus need to go through the data multiple times before producing the final model. Generally, Evolutionary Computation techniques are considered computationally expensive. But XCS utilizes Genetic Algorithm (GA) as a search technique efficiently due to its linear complexity and online learning mode.

Intrusion detection is the art of identifying illegitimate and/or malicious activity into a computer system in order to gain unauthorized access, gather crucial information about the system (i.e. to compromise system's confidentiality) or simply disrupt the normal operation (i.e. to compromise integrity and availability of the system and services it is providing). The problem is difficult to address owing to the sheer amount of correlated data flowing in and out of the computer systems and ever emerging novel attacks. Intrusion detection thus intrinsically becomes a classification in data mining problem in that, we want to classify 'Bad' or malicious events or traffic patterns from the good and legitimate ones. Current intrusion detection techniques fall into one of two categories; misuse detection and anomaly detection. The former looks for well known patterns of bad behavior in network traffic and audit data, while the latter models the normal behavior of the system and flags any deviating event as anomalous.

Our work is in the context of misuse detection. We apply XCS to learn rules automatically from the labeled training data which are then used to classify the attacks in the test data. The qualities of XCS make it intuitively

best suited for intrusion detection where the classifier system is needed to learn and adapt to the time varying characteristics of network traffic quickly and identify new attacks. We develop some modifications on the XCS model to suit the intrusion detection domain and report our findings on these experiments. Our results show that the new XCS improves considerably especially on three of the four classes in the dataset i.e. Probe, DOS and U2R.

The main contribution of this paper is in extending XCS to improve its application in the field of intrusion detection. We investigate a novel potential approach to solve the network intrusion detection problem using an efficient evolutionary approach. The goal of this work is two-fold; firstly we explore the performance of the original XCS on a large dataset. Secondly, we modify XCS to best suit the intrusion detection domain to improve on its accuracy. We chose 1999 the KDD Cup intrusion detection dataset [8] that has a large number of records with non uniform class distribution, rare classes and novel test instances.

The application of data mining techniques to intrusion detection are discussed in the following section which is followed by a brief description of XCS. From Section IV we present the experiments and the obtained results. Conclusions are then drawn and future directions are discussed in Section VI.

II. Related Work

Many researchers have explored the application of non-evolutionary machine learning algorithms in the intrusion detection domain. Sabhnani et al. [14] developed a multi-classifier model using three of the nine best performing pattern recognition and machine learning algorithms such as neural networks, clustering techniques and decision trees, that they evaluated on the KDD dataset. Agarwal and Joshi [13] proposed a two stage framework to learn rule based models. The sequential covering algorithm is used to first learn highly accurate P rules indicating the presence of a class; accuracy is gradually compromised in favor of support thereby keeping the generality of rules at the expense of some false positives. In the second stage N rules representing the absence of a class are learned on the reduced dataset to remove the false positives accumulated during the learning of P rules. A cost sensitive algorithm is then used to resolve conflict among classifiers advocating towards the same class. Giacinto et al. [6] divide connection records into six categories based on different network services. Three dedicated classifiers, each relating to a different subset of the features that constitute the KDD Cup dataset, are applied on each service module. Best performing classifiers are then fused using different techniques, where the majority voting is found to outperform the others, to discriminate between attacks and normal traffic.

There is some work in the literature that explores the application of Genetic Algorithms (GA) on the intrusion detection domain. Sinclair et al. [16] deployed independent ID3 and GA based rule generation modules on an existing expert based Intrusion Detection System (IDS). The GA module is based on a traditional genetic algorithm enhanced by applying niching using a crowding technique. Rules matching to the intrusion patterns in the training dataset get higher fitness values. Hamming distance is used as a similarity measure between two chromosomes i.e. the rules generated from the network traffic using GA. Chittur [4] applied GA on 10% of the KDD dataset to evolve best fit rules based on accuracy of predicting anomalous connections from the normal. They used Ephemeral Random Constants [10] to assign weights to individual attributes in the dataset. Gomez et al. [7] used adaptive-parameter genetic algorithm with special genetic operators (gene addition and deletion) to evolve fuzzy classifiers and applied it on 10% of the KDD dataset. The rules for Denial of Service (DOS) attacks in the dataset were evolved independently of the other attacks because of the high number of instances belonging to DOS category. They showed equivalent performance to traditional machine learning algorithms in all categories and improved performance in one of the categories. There are some other work e.g. [12] that explore the application of basic GA on the intrusion detection domain.

Traditional machine learning algorithms as discussed above require training in batch mode before exposure to the test data which implies multiple passes through training data. In domains like intrusion detection it might not be desirable, especially in a real time implementation. XCS learns in online mode thus avoiding the need of multiple

passes through the training data. There is also traditional online machine learning approaches. We will show that XCS can achieve equivalent accuracy to the traditional learning methods, thus making it naturally suitable for ID domain.

Some of the work mentioned above show promise for better or similar accuracy in the intrusion detection domain using GA [2], [7], [4]. However, to the best of our knowledge, this is the first attempt to apply a genetic based learning classifier system on this domain on that scale of data size.

III. XCS Overview

XCS is a rule-based system, where each rule represents a partial solution to the problem. The typical goal of XCS is to evolve the *Population* of rules to represent a complete solution to the target problem. XCS relies on reinforcement learning (RL) to evaluate the rules in the *Population* and GA to explore the search space by introducing new rules into the system.

XCS is designed for both single-step (immediate rewards) or multi-steps (delay rewards) environments, but in the scope of classification, a single-step environment is considered.

A rule in XCS is a macro–classifier, which contains a distinct combination of the *Condition part* or body of the rule and *Class part* or decision of the rule in the *Population*. Whenever a new rule is introduced, the *Population* is scanned to check if the new classifier already exists. If not, the new rule is added to the *Population*. Otherwise, the number of actual copies of the existing rule is incremented by one.

During reinforcement learning process, the system receives a reward from the environment for each output reflecting how good a solution is. Several rules, which match the environment's input, can participate in the process of deciding a final solution. The *action set*[1] $[A]$ is then formed of the rules participated in the decision-making process. *Fitness* of these rules are re-estimated based on the accuracy of the prediction against the actual reward received from the environment. *Experience* of a rule is considered as the number of time it has been contributed to the system's decision or participated in the *action set*.

GA in XCS is responsible for introducing a new rule into the population. During the selection process, two parents from the *action set* $[A]$ are selected with probability proportional to their fitness. Two offspring are generated by reproducing, crossing–over, and mutating the parents. Parents continue to stay in the population competing with their offsprings. If the population size is less than a certain number, offsprings are inserted into the population; otherwise, two of the most inaccurate classifiers are deleted from the population before the offspring can be inserted.

Following is a very simple description of XCS working:

1) Receive an instance from the environment
2) Scan through the *Population* set for rules which match the input instance
3) IF *Population* set is empty OR no matching rules are found from the previous step
    a) Create random rules but match the input instance for each possible action
    b) Initialize parameters to default values for the rules
    c) Insert the rules into the *Population* set
4) Form a *match set* of all matching rules
5) Send the action of rules in the *match set* with highest prediction value to the environment
6) Form an *action set* of all rules advocating to the same output action in the *match set*
    a) Update parameters of all rules in the *action set* based on the reward received from the environment
    b) Invoke GA on the *action set*

[1]The *action set* is the subset of the *match set* used to give the system response (class)

TABLE I

CATEGORIES OF DATA INSTANCES AND THEIR PROPORTION IN TRAINING AND TESTING SETS OF KDD DATASET

Category	% in training data	% in testing data	Example
NORMAL	75.6111	19.4815	Normal
DOS - Denial Of Service	1.2893	73.9008	SYN flood, Smurf
PROBE	23.0017	1.3394	IP sweep, Port scans
U2R - User to Root	0.0048	0.0733	Buffer overflow
R2L - Remote to Local	0.0929	5.2050	Password guessing

7) Repeat from 2 for every new instance

IV. EXPERIMENTAL SETUP

We use the dataset provided for the 1999 KDD Cup which is developed on the dataset used for 1998 Intrusion Detection Evaluation (IDEval) program organized by MIT Lincoln Labs in coordination with DARPA and AFRL [2] [11]. During MIT Lincoln Lab IDEval program, seven weeks of raw data containing simulated attacks and background traffic was provided along with the audit logs collected at different hosts of the simulated network for training IDSs. Trained IDSs were then evaluated on two weeks of test data containing new type of attacks in addition to the attacks in the training dataset. Table 1 shows the categories and their proportion in training and testing datasets.

Data was pre-processed in a similar manner as described in [14]. Different attack types are mapped to four categories from 1 to 4 with Normal class mapped to 0. All symbolic attributes such as protocol type and service were also mapped to an integer value from 1 to the maximum number of corresponding symbols. All continuous attribute values were then scaled linearly from 0 to 1; very large values (source and destination bytes) were scaled logarithmically.

We used the default parameters values used by Wilson [19] for XCS. All experiments presented are averaged over 30 independent runs.

V. DISCUSSION AND RESULTS

A. The Traditional XCS

Table II presents the prediction accuracy of the conventional XCS on the KDD dataset. The table shows that conventional XCS gets high accuracy on the R2L class, lower accuracy on class Normal and close to 0% accuracy on all other classes.

TABLE II

MEAN AND STANDARD DEVIATION OF PREDICTION ACCURACY OF CONVENTIONAL XCS IN 30 RUNS

Class Prediction	Accuracy
NORMAL	0.572 ± 0.250
DOS	0.000 ± 0.000
PROBE	0.000 ± 0.000
U2R	0.002 ± 0.005
R2L	0.637 ± 0.293

In the testing phase, XCS predicts the class for new testing instances based on the current knowledge. When XCS confronts with an absolute unseen instance, several random but matching rules are created and inserted in the *Population* set. Traditional XCS applies exploration and exploitation alternatively for training and testing and

[2]Air Force Research Laboratories

therefore these new inserted rules are verified at some points later. In order for XCS to fit into the intrusion detection domain, we decided to train the system at once and then explore the latest model on the test set. As the result, rules inserted in the population at the testing phase have no chance to be verified. Thus they are not good enough for using in system prediction. Therefore, the decision of unseen instances is then based on the default class, which is R2L in this case.

The prediction accuracy of the Normal class is higher than other classes but it is still very low if we consider the training samples comparing to other classes. There are 812,813 training instances for Normal class, which is about 75% of the training set. Since the model is trained mostly on Normal class for a substantial time, we should expect it to be able to at least correctly classify Normal class. As described before, several rules are created and inserted in the *Population* set in response to unseen instances. Since the population is fixed in size, some rules are deleted to make room for the new rules. The candidate rules being selected for deletion would likely belong to the Normal class as the model would have a large portion of rules in this class due to the large number of training samples. These rules are normally evaluated well in the training process. Hence, replacing them by several random rules drives the system to work as a random search.

B. The Modified XCS

This sub-section points out several reasons for which the conventional XCS fails on large dataset in general and the KDD dataset in particular. We also propose several ways to overcome each problem and in the end present the result of our modified XCS.

1) The Mutation effect: XCS uses GAs to explore the search space with an attempt to evolve a population of rules which represent a complete action-map of the problem space. The mutation operator in GA plays an important role in searching since it helps to escape from local optima and creates opportunities for new rules to arise in new areas of the search space.

Each rule has two components: the body and the predicted class. XCS uses macro-mutation, where it mutates every attribute of a rule's body and rule's class at a certain rate. Mutating the body of a rule generalizes it beyond its locally covered area. It is important to emphasize here that the mutation rate is usually small and mutation can be seen as a local search operator. Hence, by changing one of the conditions in a rule, the rule is somehow generalized to cover other areas of the search space in the neighborhood of the original rule.

XCS also mutates the predicted class. This is somehow less intuitive because: (1) if only the predicted class of a rule is mutated, it will increase conflict within the rule set; and (2) if the condition component is also mutated, it is more likely that by generalizing the rule to cover its immediate neighborhood, the class will remain unchanged. Overall, we believe in many real life problems, allowing the rule to mutate the predicted class will cause harms to the classifier most of the time. Hence, we decided to apply mutation only to the body of the rule. In so doing, we still allow the system to explore a new area without disrupting the rule.

2) Allowing the Deletion operator to handle minority classes: The population of XCS is fixed, and the user decides in advance an appropriate population size for the problem in hand. When a new rule is created (e.g. a product of GA), it is inserted into the population. If the maximum allowed population size is exceeded, an old rule needs to be deleted. Butz and Wilson [3] described the deletion procedure as choosing individuals (for deletion) by roulette-wheel selection. The deletion vote of each rule is based on the size of the *action set* and the fitness value of each rule in the population. Moreover, if a rule has participated in the system decision-making for a sufficient numbers of time and its fitness is significantly lower than the average fitness in the population, the vote to delete it is increased further.

Testing XCS on the KDD dataset, we found out that the final population is dominated by rules for the Normal class rather than abnormal classes (4 attack classes). This was because of the sample distribution in the KDD dataset which has a high number of Normal instances constituting approximately 75% of total records. Thus the system spends more time training instances of Normal class than any other class. Consequently, GA is more likely to explore rules for predicting the Normal class and as a result ending up with a population having more number of rules for the Normal class. Since these rules have high chance of getting evaluated more number of times than rules for other classes, they become more accurate in predicting the reward received from the environment. Therefore, their fitness is higher than any other rules (low fitness rules for the Normal class are eliminated early), which implies that the rules for the Normal class have a much lower chance to be deleted compared to rules for other classes with the current deletion scheme. This explains why rules of abnormal classes do not remain in the final population.

We decided to include the class distribution factor of the current population into consideration during the deletion scheme. Before deleting a rule, we estimate a class distribution of the population based on the amount of instances of each class the system received up to this point, the more rules predicting to a class in the population, the higher the chance of their being eliminated. Hence, we maintain the diversity in the population despite the fact of unbalanced sample distribution in the training set.

3) Distance Metric: As being mentioned in the previous section, in order to apply XCS to the KDD dataset, we modified the traditional XCS by first training it on the training dataset, then using the obtained model to predict the attacks in the test set.

Traditional XCS is trained and tested alternatively. In both cases, the covering technique is utilized if the system fails in recognizing an input. In the modified model, we decided not to call the covering technique for unseen instances in the testing phase. It is because the model does not learn in the testing phase and therefore doing it can result in deleting useful rules and leaving the population with random rules.

We come up with a distance metric instead to choose rules which are close to the unseen input. The distance is measured based on the Euclidean distance from the input to each rule in the population in all dimensions (41 attributes). We introduced a new parameter called DISTANCE_THRESHOLD to decide when a rule can be called *close match* the input. The rules with a Euclidean distance less than this threshold are considered to *close match* the input. The decision of the system is then based on voting between these *close match* rules.

In order to make a compact and more precise population, we also decided to post process the final population obtained from the training phase before using it for classification in the testing phase. In this process, we filter out rules with no or less *experience*. It is believed that the more *experience* a rule has, the more it was evaluated or learned from the environment's feedback, and therefore the more accurate it becomes. Remember the fitness of XCS is based on the reward prediction (I), and therefore accuracy in this sense means predicting the reward received from the environment. Normally, rules without evaluation by interacting with the environment might mislead the decision of the system and therefore it is inefficient to keep these rules.

C. The Result of the Modified XCS

We have tested each of our modifications separately on the conventional XCS and obtained fairly improved results. However, the results did not fully satisfy us. We then decided to put all modifications together and the result is reported in this section.

Table III shows the prediction accuracy of the modified XCS for normal and attacks. It can be easily seen that the modified XCS outperforms the conventional XCS in all classes except the R2L class. As explained in the previous section, the high accuracy on R2L class is achieved due to default class for unseen instances. Therefore, the prediction accuracy of the modified XCS is lower than the conventional XCS, but it is based on the system's

TABLE III

MEAN AND STANDARD DEVIATION OF PREDICTION ACCURACY IN 30 RUNS

Class Prediction	Accuracy
NORMAL	0.957 ± 0.043
DOS	0.491 ± 0.229
PROBE	0.930 ± 0.144
U2R	0.085 ± 0.121
R2L	0.039 ± 0.022

experience not on the default value.

Hence, we proposed several improvements for XCS to deal with a large and bias dataset. The result shows that the modified XCS has improved over the traditional XCS in all classes.

We have calculated the cpu time for running the original and modified XCS. During the initial training phase, XCS was able to process 39,000 instances a minute while during the testing phase (no GA takes place), it was able to process 46,800 instances a minute. There was no differences in the runtime between the original and modified one. These calculations show the power of XCS and the potential to break the barrier to finally adopt evolutionary techniques for real-life data mining problems.

VI. CONCLUSIONS

In this work we have explored the application of ELCS on a large dataset. We chose the domain of intrusion detection which is becoming an increasingly important problem in computer security. This paper showed that conventional XCS does not perform as well as it was expected to be in the intrusion detection domain. We scrutinized the algorithm and identified several reasons for its inferior performance. We then modified XCS to increase the detection accuracy by introducing heuristics like nearest neighbor. We update the rule population by deleting the rules proportional to the class distribution seen by the system thus making sure that rules covering rare classes are not deleted. The modified XCS performs considerably better than the original XCS.

Most intrusion detection systems implemented on actual computer systems to date are signature based which have the limitations of missing novel attacks. On the other hand implementation of anomaly based systems is still in its premature state with the limitations of producing high number of false alarms and a delay in the detection of the actual attack due to processing time of anomaly detection system. As discussed XCS' biggest advantage is its ability to work in online mode, where it can update the model continuously based on the feedback from the environment. The GA module in XCS evolves a population of accurate rules thus giving diversity and a better chance of detecting unseen attacks. We were able to handle a traffic during the test phase up to 46,800 instances a minute. This is a remarkable performance for an evolutionary-based system. Nevertheless, to use the system in real life, we will have to increase the speed with an order of magnitude. There are a number of ways to do that. One way is to implement XCS in a distributed mode where different module will cover certain types of attacks by incorporating domain knowledge in the rule generation. This is a focus of our current on-going research in this area.

VII. ACKNOWLEDGEMENTS

Work reported in this paper was funded by the Australian Research Council Linkage grant number LP0453657 and University College Postgraduate Research Scholarship.

REFERENCES

[1] J. Bacardit and M. V. Butz. *Data Mining in Learning Classifier Systems: Comparing XCS with GAssist*. Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, June 2004. IlliGAL Report No. 2004030.

[2] E. Bernadó, X. Llorà, and J. M. Garrell. XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks. In *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS-2001)*, pages 337–341, 2001.

[3] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. In *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 253–272. Springer-Verlag, 2001.

[4] A. Chittur. Model generation for an intrusion detection system using genetic algorithms. High school honors thesis, NY Ossining, 2002.

[5] P. W. Dixon, D. Corne, and M. J. Oates. A preliminary investigation of modified XCS as a generic data mining tool. In *Advances in Learning Classifier Systems: 4th International Workshop, IWLCS*, pages 133–150. Berlin Heidelberg: Springer-Verlag, 2001.

[6] G. Giacinto, F. Roli, and L. Didaci. A modular multiple classifier system for the detection of intrusions in computer networks. In *Multiple Classifier Systems*, pages 346–355, 2003.

[7] J. Gomez and D. Dasgupta. Evolving fuzzy classifiers for intrusion detection. In *2002 IEEE Workshop on Information Assurance*, June 2002.

[8] S. Hettich and S. D. Bay. The uci kdd archive. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[9] J. H. Holland. Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine Learning, an Artificial Intelligence Approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.

[10] K. JR. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[11] R. P. Lippmann and M. A. Zissman. 1998 darpa/afrl off-line intrusion detection evaluation. dataset available at http://www.ll.mit.edu/IST/ideval/data/data_index.html, 1998.

[12] M. M. Pillai, J. H. P. Eloff, and H. S. Venter. An approach to implement a network intrusion detection system using genetic algorithms. In *SAICSIT '04:*, pages 221–221, , Republic of South Africa, 2004. South African Institute for Computer Scientists and Information Technologists.

[13] A. Ramesh and J. V. Mahesh. PNrule: A new framework for learning classifier models in data mining (a case-study in network intrusion detection). In *1st SIAM Conference on Data Mining.*, April 2001.

[14] M. Sabhnani and G. Serpen. Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context. In *MLMTA*, pages 209–215, 2003.

[15] S. Saxon and A. Barry. XCS and the Monk's problems. In *Learning Classifier Systems, From Foundations to Applications*, pages 223–242, London, UK, 2000. Springer-Verlag.

[16] C. Sinclair, L. Pierce, and S. Matzner. An application of machine learning to network intrusion detection. In *15th Annual Computer Security Applications Conference, 1999. (ACSAC '99)*, pages 371 – 377, 6-10 Dec. 1999.

[17] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[18] S. W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[19] S. W. Wilson. Get real! XCS with continuous-valued inputs. In P. Lanzi, W. Stolzmann, and S. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications, LNAI-1813*, pages 209–219, Berlin, 2000.