Applying Accuracy-based LCS to Detecting Anomalous Database Access

Suin Seo
Dept. of Computer Science
Yonsei University
Seoul, Korea
tndls9304@yonsei.ac.kr

Sung-Bae Cho
Dept. of Computer Science
Yonsei University
Seoul, Korea
sbcho@yonsei.ac.kr

ABSTRACT

Database intrusion detection (DB-IDS) is the problem of detecting anomalous queries in transaction systems like e-commerce platform. The adaptive detection algorithm is necessary to find anomaly accesses when the environment changes continuously. To solve this problem, we used accuracy-based LCS (XCS), one of the primary model of adaptive machine learning method, for detecting malicious accesses in databases. In the problem of database intrusion detection which changes the detecting targets, we found and analyzed the patterns of rule generation to show systemically how the adaptive learning of XCS algorithm is working in practical usage.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**; • **Security and privacy** → Intrusion detection systems;

KEYWORDS

Database intrusion detection system, Accuracy-based learning classifier system, Analysis of adaptation processes

ACM Reference Format:
Suin Seo and Sung-Bae Cho. 2018. Applying Accuracy-based LCS to Detecting Anomalous Database Access. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan,* Hernan Aguirre (Ed.). ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3205651.3208315

1 INTRODUCTION

For years, the amount of data we are dealing with sharply increased as using information techniques in everything in our lives. All valid information is stored in the databases and database management systems (DBMS) are managing and utilizing them efficiently in every domain. Among several types of DBMS, the most used DBMS is relational database management system (RDBMS), that changes and updates many databases in the world through standard query language (SQL). However, the databases used in the large platform are always being exposed to threats of abnormal queries which execute illegal and harmful transactions. Database intrusion detection system (DB-IDS) is the system that detects these anomalous queries and protects the database from the threats. Unlike other domains of intrusion detection systems (IDS), DB-IDS is hard to build for the several reasons. First, almost attacks to RDBMS is from the inside, not the outside [17], and internal intrusions evolve more complexly than the external ones. Also, internal attacks that are processed by the authenticated users make the unexpected patterns which are far from the standard [11].

In the system like e-commerce environments, a vast amount and various kinds of transactions occur in every second. It is difficult that modeling abnormal transactions rather than modeling normal one in this case since there would be few or no anomalies in overall data. This class imbalance condition occurs when modeling IDS in the specific domain. Therefore, we generate the model extracting the characteristics from only of regular queries for classifying the anomalous queries even if there is no anomaly in training samples.

We use TPC-E benchmark [5] for making the virtual dataset of queries about e-commerce systems, which have entire 11 transaction roles of users such as broker, customer, trade, and so on. Then we regarded a query as an outlier if the query doesn't belong to any pre-defined ruleset. For distinguish the query which is non-matched with role-query or not categorized in entire 11 roles, the model should be able to distinguish the proper roles of the given query. Every object in the transaction schemes has authorities which limit and control their work. In this point of view, machine learning method is applicable to classify the given queries as pre-defined roles. Because the characteristics extracted from the query include the attribute of the events, the algorithm can detect whether the transaction is invasive by the patterns of the query [3]. However, general machine learning techniques are fundamentally static learning one which cannot detect the object which changes over time, also about first-seen attacks. This aspect is because the classifier model is created after inputting the entire data; thus, we need the model which trains and tests the data simultaneously.

In this paper, we use accuracy-based LCS (XCS) [20] for showing that the adaptive process of XCS is working well in DB-IDS problems. Learning Classifier System (LCS) [7], the primary model of XCS, is easy to analyze internal structures as well as to implement and has the advantages of both machine learning approach and genetic algorithm by combining both of methods. We experimented XCS for classification tasks in the cases that the set of target classes changes for the several phases with the simulated data of TPC-E transaction benchmark. The results show us detailed processes about the classifiers update their rules, and behavior patterns when the target actions vary, for the various cases of environmental changes.

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208315

2 RELATED WORKS

2.1 Machine Learning for Intrusion Detection

Intrusion detection system (IDS) divides into two kinds of major approaches: Signature-based detection, and Anomaly-based detection. Signature-based, one of the leading methods of IDS, does not adaptively work since it detects the intrusions by looking for specific patterns in the dataset, for example, byte sequences found in network traffic and malicious accesses. Therefore signature-based IDS is mostly used for anti-virus software in PCs because of the system is efficient for known attacks although the databases update is essential. However, not applied in DB-IDS problem, because, notably, in RDBMS domain, the signature-based method is inherently susceptible to evasion methods that take advantage of the expressiveness of the SQL or alternate character encodings [15].

On the contrary, anomaly-based IDS creates regular profiles by training normal traffics; then detects intrusions by measuring the deviation of new traffic from the pattern of the generated model [1]. There are many pieces of research which have been studied IDS using data-driven machine learning techniques, such as association rules [6] Bayesian network [9] and hidden Markov model [2]. In recent, the methods expanded as the Gaussian mixture model, AdaBoost [8], and random forest [16]. However, these static learning algorithms are not helpful to detect the attacks when the types or patterns of target change.

There are several methods to work adaptively for the domain for solving intrusion detection problem. At first, very widely used adaptive algorithm is the genetic algorithm (GA) which imitates the heredity system of a living organism [12]. GA is utilized at not only directly applied in the method but another learning algorithms like decision tree classifiers [18].

2.2 Preprocess of Queries

2.2.1 Query Log Parsing. A query used in RBDS follows the grammar of SQL. Therefore, the features of queries are inherent in each part of grammatical elements. The parser divides the query into several phrases depending on its clauses and kinds of the main words, reserved words in SQL, reflecting the characteristics of the query.

2.2.2 Feature Extraction. The feature extractor extracts the feature values from each parsed part. In this process, the extractor output the vector Q about several elements of the input that have the characteristics of a query.

$$\begin{aligned} Q = &\{\text{SQL-CMD[]} + \text{PROJ-REL-DEC[]} + \text{PROJ-ATTR-DEC[]} \\ &+ \text{SEL-ATTR-DEC[]} + \text{ORDBY-ATTR-DEC[]} \\ &+ \text{GRPBY-ATTR-DEC[]} + \text{VALUE-CTR[]}\} \end{aligned} \tag{1}$$

2.2.3 Feature Selection. We use Information Gain (*IG*) and Gain Ratio (*GR*) to select the appropriate features. *IG* is the difference between entropy I of a prior state and the present one.

$$I(S) = -\sum_{k=1}^{K} \frac{|s_k|}{|s|} \log \frac{|s_k|}{|s|} \tag{2}$$

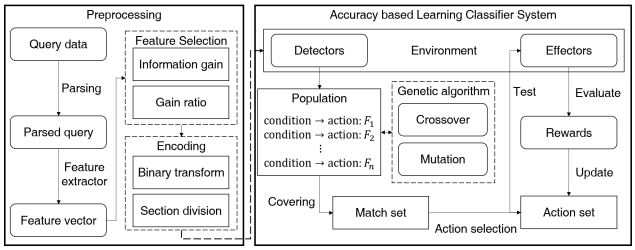

Figure 1: Overall architecture of DB-IDS system using XCS algorithm with preprocessing of query parsing.

$$IG(Y) = I(S) - \sum_{m=1}^{M} \frac{|s_m|}{|s|} I(S_m) \tag{3}$$

Where s_m is the number of queries in outcome $m \in M$. And GR is the ratio between Information Gain and Information Value (IV).

$$IV(S|Y) = -\sum_{m=1}^{M} \frac{|s_m|}{|s|} \log \frac{|s_m|}{|s|} \tag{4}$$

$$GR = IG/IV \tag{5}$$

$$Merit = GR - IG \tag{6}$$

Then we choose the number of features which have the maximum difference of average merit of *IG* and *GR* in equation (6). As shown in the chart of Fig. 2, 14 is chosen by this criterion for the index of maximum value. We sort the features in descending order of the calculated *IG* and select the 14 features which are from the top.

3 PROPOSED METHODS

3.1 Adaptation Process of XCS

XCS is the classifier system that adopts the rule representation of Michigan approach which represents the pattern of a single rule in an individual classifier and places a priority on the reinforcement learning [20].

3.1.1 Rule Representation. The rule of LCS (including XCS) expresses the relation state and predictions. The relations have the form of {IF: THEN} expression, same as condition → action. Given binary data, the rules of the LCS follow the ternary format: $\{0, 1, \#\}^l$, where l is the length of the input data, and '#' means "don't care." Every condition has action represented as an integer, and every rule has the weights called fitness. Each condition-action pair constructs a classifier, which acts as decision boundary. For example, if the rule of LCS is 00###|##000|000#0 and the range of each variable, x is 0 to 1, the condition indicates that each variable belongs to the range of $0 \le x_1 \le 0.8$, $0.8 \le x_2 \le 1$, and $0.2 \le x_3 \le 0.4$. Then from this decision boundary, LCS predicts the value (action), and fitness, the weights of the rule. These classifiers compose the population [P] of LCS by grouping themselves.

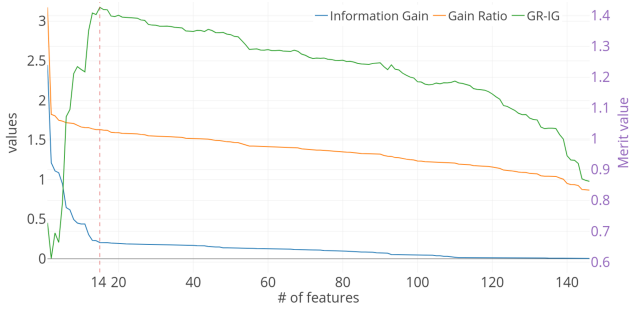

Figure 2: Line plots of IG, GR, and GR-IR for merit, which is highest at the number of features is 14.

Table 1: Variables and its characteristics selected by equation 6. The elements in "Type" column means C (categorical), I (integer) and B (binary). The encoding length of queryLength is manually selected since the type is continuous numeric.

Variables	Type	Min	Max	Encode
queryMode	C	1	4	4
queryLength	I	36	687	custom(11)
fieldNum	I	0	16	8
tableNum	I	0	10	6
projectionNum.customer	I	0	10	6
projectionNum.trade	I	0	10	4
projectionId.zip_code	I	0	7	1
whereClauseId.watch_list	I	0	3	4
orderById.customer	I	0	2^{23}	1
groupByNum	B	0	1	1
groupByNum.broker	B	0	1	1
groupById.customer	I	0	2^{23}	1
stringValueNum	I	0	5	6
numericValueNum	I	0	7	6
Overall	-	-	-	61

3.2 Component of Model

The population generates match set [M], whose conditions are satisfied, for the input data. If the number of [M] is less than the parameter, the system executes covering process within limit of maximum population N. The process creates a new classifier composed randomly selected matched current condition and pre-labeled action, among those not included yet in [M]. Updated match set determines which action to select. There are two ways to choose the action. One is selecting the action having the highest average of prediction, and the other is selecting the random action for exploring the search space. The probability of random action is chosen by the parameter P_{exp}.

3.3 Fitness Updating Process

At first, the environment returns rewards r, which results the classifier gets the highest score for the matched action with the rule. Then the prediction p is adjusted by the difference between prediction and rewards.

$$p \leftarrow p + \beta(r - p) \tag{7}$$

The closer p is to r, the higher accuracy of the rule; β is the learning rate. After p is updated by equation 7, we update the prediction error.

$$\epsilon \leftarrow \epsilon + \beta(|r - p| - \epsilon) \tag{8}$$

Error ϵ converges the difference between reward r and prediction p therefore the more accurate prediction of the classifier, the lower error. We calculate classifier's accuracy with the error as follows.

$$\kappa = \begin{cases} 1 & \text{if } \epsilon < \epsilon_0 \\ \alpha(\epsilon/\epsilon_0)^{-\nu} & \text{otherwise} \end{cases} \tag{9}$$

Parameters of equation 9 such as α, ϵ_0, and ν control the amount of decline in accuracy when the classifier is wrong. We compute the relative accuracy κ' dividing the κ by the total accuracies of actions in the action set and update the fitness F for the κ'

$$\kappa' \leftarrow \frac{\kappa}{\sum_{x \in [A]} \kappa_x} \tag{10}$$

$$F \leftarrow F + \beta(\kappa' - F) \tag{11}$$

Through the above processes, fitness F converges to modelâĂŹs relative accuracy. Therefore, we can evaluate each ruleâĂŹs quality by the fitness F.

Table 2: Eleven standard transactions in TPC-E benchmark. Each transaction type has the allowed query type and authority for updating database.

No.	Transactions	Available query	Authority
1	Broker-volume	SELECT only	Read-only
2	Customer-position		
3	Market-watch		
4	Security-detail		
5	Trade-status		
6	Trade-lookup		
7	Trade-order	SELECT/INSERT	Read/Write
8	Trade-update	SELECT/UPDATE	
9	Data-maintenance		
10	Market-feed	SELECT/INSERT/ UPDATE/DELETE	
11	Trade-result		

3.4 Use of Genetic Algorithms

After the fitness is updated, the genetic algorithm modifies the rules in action set. The applied algorithms are crossover and mutation. Crossover mixes two conditions at specific points. The algorithm selects two parent rules in the action set which have the same prediction, then, generates the children by swapping the parts of each parents' rules from one point to another. Applying crossover algorithm is controlled by crossover probability χ.

Another algorithm is the mutation. The algorithm is a method of transforming a part of the rules. Among the conditions in action

Table 3: Scenarios used in the experiments. Each phase includes target classes. Each scenario is about detecting roles when they are increasing, decreasing, and switching.

Scenario	Phase1	Phase2	Phase3
Scenario A	[2,4,8]	[2.4,8,10]	[2,4,7,8,10]
Scenario B	[1,5,6,7,9]	[1,5,7,9]	[1,5,9]
Scenario C	[1,3,5,7,9]	[2,4,6,8,10]	[3,5,7,9,11]

set, the algorithm creates a new condition by replacing the wildcard of the selected one with real value (0 or 1) while maintaining the matched condition. Applying mutation is controlled by mutation probability μ. Genetic algorithms help XCS to generate more various rules, exploring available search spaces, even if the most of new rules are fail for alive in the environment.

3.5 Applying XCS to Query Features

Total 191 attributes are extracted from the raw query by feature extraction process in Section 2.2, selecting the number of features as 14, and Table 1 shows the list of the variables. Entire encoding length of a query is 61.

Encoding process transforms each vector of selected values as a string of ternary $\{0, 1, \#\}^l$. The length of encoded strings is automatically determined by calculating information gains only about the training data, 80% of entire data, not about the rest (the test data). Making decision boundary concerning training data is not affected by the classification performance even if the same encoding process apply to the test data. Processed query features are transformed to the required format which represents the numeric values as categorical, same with one-hot encoding process. In case of binary format, we assign a single character, which could express bit information 0 or 1 rather than represent one-hot encoding with the length of 2 to prevent assigning redundant encodings.

4 EXPERIMENTS

4.1 Experiment Settings

4.1.1 Benchmark Database. We use TPC-E database benchmark for simulating online transaction processing (OLTP) workload of a brokerage firm [5]. In this schema, we use 11 standard transactions in generated 11000 queries for 1000 for each role in Table 1.

4.1.2 Scenario Settings. To analyze the rule generation steps when the classification target changes, we set several scenarios which have the different aspect of change. The situations are expressed in Table 3, and in short, we considered the case of when the classification target increases, decreases, and switches (maintaining the number of classes).

4.1.3 Conducting Experiments. We divided training and test data as 8:2 and get the reward for training data and calculated classification accuracy for test data for every training epoch of entire training data. For every training epoch, we inputted the training data to the classifier to find out whether the system is precisely adapted.

Table 4: The parameters of XCS which we used in the experiments.

Parameters	Symbol	Value
Maximum population	N	2000
Learning rate	β	0.15
Accuracy coefficient	α	0.1
Accuracy power	ν	5
Error threshold	ϵ_0	0.01
Crossover probability	χ	0.75
Mutation probability	μ	0.03
Deletion threshold	θ_{del}	2
Wildcard probability	$P_{\#}$	0.1
Exploration probability	P_{exp}	0.2

Table 5: Evaluation information at epoch 40, 41, and difference between them. The rules in epoch 41, are not only having fewer rules, also quite more precise than the that in epoch 40.

Point	Train accuracy	Test accuracy	# of rules	Avg. fitness
epoch 40	0.7044	0.7238	477	0.1754
epoch 41	0.7628	0.7275	433	0.1835
difference	**+0.0584**	**+0.0037**	**-44**	**+0.0081**

The first experiment uses scenario A, and training process continued to 30 epochs until showing the overfitting or limit of rule generation. With second scenario B, same with the scenario A, the experiment is conducted that one phase experiences 30 epochs of training. In the case of last scenario C, because the size of training data is higher than scenario A and B, we trained 50 epochs in one phase rather than 30.

4.1.4 Selecting Parameters of XCS. There are many parameters, which set manually, for XCS method. The parameters for the experiments are listed in Table 4. Most of the parameters have been manually tuned, having no additional meanings, except, deletion threshold θ_{del} which is the patience step of rules with no matched conditions. We set the θ_{del} to a small value since we want the XCS to adapt fast and dispose of meaningless rules as soon as possible.

4.1.5 Evaluation of XCS. We evaluated the rules in XCS for two metrics. One metric is training/testing accuracy which means that the XCS rules are generated and fit well in the task environments. Training accuracy is evaluated instantly as the rule is updated for the data, and the test accuracy is evaluated in every step after one epoch of training is over to check how well the model adapts to the data.

The other metric is fitness to evaluate the quality of rules. Although fitness of the rule is the relative value, we confirmed the fitness being higher as the model is being converged. Absolute accuracy κ is another option but we didn't use, because whenever generating a new rule, κ is set to 1 by initial error ϵ_0 which overestimate the model.

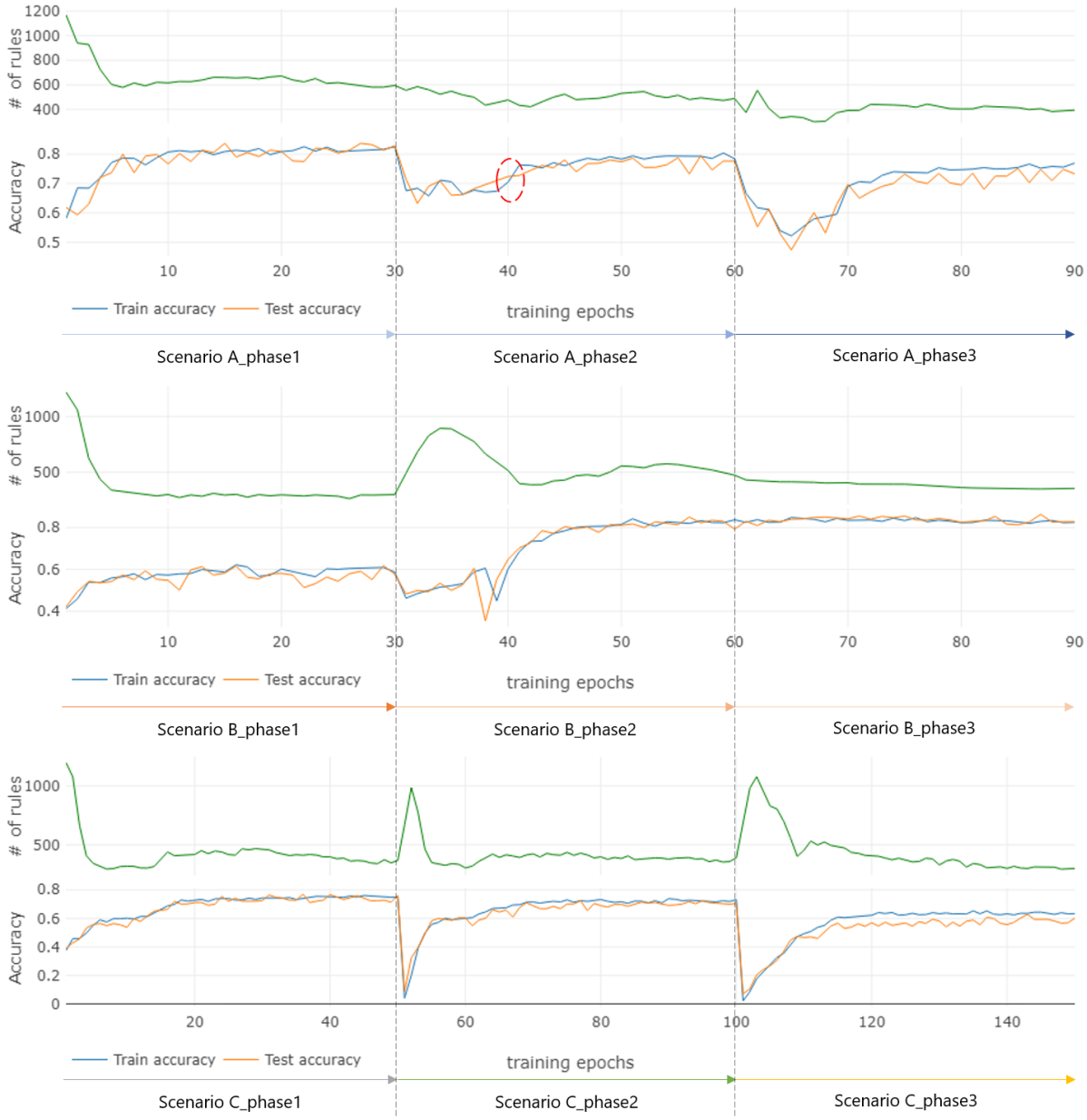

Figure 3: The number of rules, and train, test accuracy in Scenario A, B, and C. Red circle is the point which we analyze at focusing adaptation processes.

4.2 Results

Fig. 3 showed the number of rules and the accuracy of training and test data when the system finished the training step at each scenario. The figures show XCS's adaptability by showing the convergence patterns as the learning step proceeds. In the scenario A, the step between epoch 40 to 41 sharply increases their performance, details in Table 5. We investigated the difference between these points which not only increased the accuracy over 5%, also decreased the number of rules.

4.3 Focusing on Adaptation Process

4.3.1 Performance Improvement. The accuracy of the model adaptively increased in training steps of each phase in every scenario. Especially, the red circle in Fig. 3 is the point whose accuracy is increased rapidly, almost 5%, besides the number of rule decreases, only in one step. Thus, we focused on these points and summarized what happens and effects to the performance.

The left plot of Fig. 4 shows that the top 30 classifiers' fitness of the model in epoch 40 and 41, confirming that fitness of classifiers

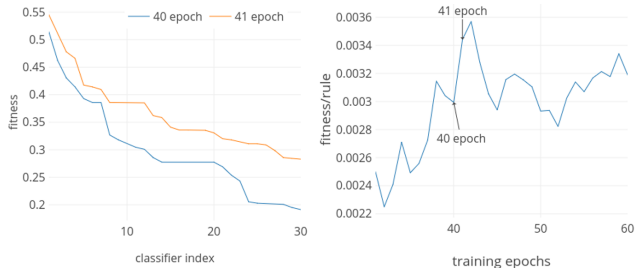

Figure 4: Top 30 fitness of rules in epoch 40-41 (left), and "Fitness / # of rules" in training phase2, Scenario A (right).

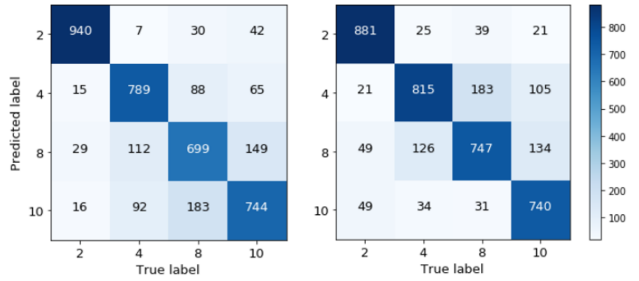

Figure 5: Confusion matrixes of classifier models at epoch 40 (left) and epoch 41 (right) for entire dataset of class 2, 4, 8, and 10 used in phase2 of Scenario A.

mainly increased in that period. Whereas, the right chart of Fig. 4 expresses the approximate fitness value assuming one rule virtually has. The fitness per rule is increased by the training step proceed. This value means the classifiers are optimized for the data, and higher fitness per rule result that the system would classify more accurate only with the fewer rules than the case of the value is low.

Fig. 5 represents the confusion matrixes of XCS models whose training step epoch is each 40 and 41 in phase2 of scenario A. As the figure shows, we can verify the classification performance is remarkably better at epoch 41 than 40. The accuracies of epoch 40 and 41 are each 79.3 and 86.25 which slightly differ from the accuracies in the training step since the target data is entire dataset (training data + test data) for showing how the model fit as well in the whole dataset.

The improvement points are just the higher classification performances of rule 4 and rule 8 as shown in the differences of the matrices in Fig. 5. Could we conclude that the differences between epoch 40 to 41 are just generation of effective rules for class 2 and 4 (or deletion of meaningless rules for class 2 and 4)? So, we tried analysis about actual classifiers' action and their fitness values which are changed in that step.

4.3.2 Rule Analysis. Not only for seeing the adaptation of the XCS algorithm, but also for finding the differences of rules in the classifier while training, we make the indexes of the classifiers to check which rule is selected or not. Then, generate a set of the rules, then investigate the result subtraction and intersection of classifier sets.

Table 6: Fitness per rules in epoch 40, 41, and between them (diminished, updated, maintained). Entire fitness per rule is not very changed since generated rules have low fitness than others. Nevertheless, we observed large fitness improvements of maintained classifiers, regardless of higher accuracy in that step. Entire fitness is just average of four fitness values for each action.

Categories	Actions	# of rules	Avg. fitness per rule
set40	2	51	0.155
	4	57	0.150
	8	221	0.177
	10	148	0.175
	entire	477	0.164
set41	2	55	0.166
	4	47	0.150
	8	165	0.172
	10	166	0.171
	entire	433	0.165
set40 (diminished)	2	26	0.145
	4	29	0.144
	8	142	0.177
	10	66	0.182
	entire	263	0.162
set41 (updated)	2	30	0.138
	4	19	0.112
	8	86	0.136
	10	84	0.151
	entire	219	0.134
set40 ∩ set41 (maintained, epoch 40)	2	25	0.166
	4	28	0.155
	8	79	0.178
	10	82	0.170
	entire	214	**0.167**
set40 ∩ set41 (maintained, epoch 41)	2	25	0.199
	4	28	0.169
	8	79	0.211
	10	82	0.402
	entire	214	**0.245**

In the first four large rows in Table 6, even if the age of updated rules is one, the fitness values of classifiers are not overwhelming the previous ones, besides, weaker than the residential rules. From this result, we conclude that the rapid increase of the accuracy is by existed classifiers from before not the newly generated ones. Therefore, we analyzed the accuracy for just remained rules in epoch 40 to 41 on the last two rows in Table 6.

Contrary to expectations, there is a huge improvement of fitness in classifiers for class 10, small about class 4 and 8 (although fitness values of every class increase). From these observations, the rapid growth of the performance is from updating the importance of existed rules, not the new rules, and throwing away the wrongly produced classifiers. Mainly, there is the considerable progress in fitness values of classifiers which classify class 10, but the actual

accuracy has no improvement as shown in Fig. 4. However, a precision enhancement of 0.719 (epoch 40) to 0.867 (epoch 41) at class 10 indicates that the updated classifier affects the categorization of other classes (4 and 8) without no doubt.

4.3.3 Cases of Scenario B and C. We tested Scenario B whose the classification target decreases, but the system discards well the useless classifiers as soon as possible automatically by small deletion threshold θ_{del}. In the former steps of phase 2 in Scenario B, generating rule process occurred, but is for improving accuracy with exploration and genetic approach. In Scenario C, as we expect, the accuracy is hitting bottom when the phase is changed. For handling this strange situation, the system sweeps the existed rules and generates many rules for adjusting to new environments. There is no relation between classifiers in phase1 to phase2 and phase2 to phase3 since the characteristics of the encoded feature string are different for the classes.

4.3.4 Saturation of Classifier Rules. In every scenario, because the maximum population N is previously set before the training step, XCS has the inherent limitation of storing information of the input data. These characteristics disturb rule generation and optimization when there are many kinds of class or are large data which contains the intricate pattern. Since the more various and complicated patterns of the dataset, the more classifiers need, so that the number of rules affects to the model's convergence point. By looking the convergence points of each phase of Scenario A and C in Fig. 3, there is the weak relationship between the number of classes and the convergence accuracy, the more the class, the less the convergence accuracy. This intuition gives us that XCS is not fit the dataset which has the considerable number of actions.

5 CONCLUSIONS

In this paper, we applied XCS for intrusion detection on database transactions for investigating the adaptation principles of XCS. The empirical experiments show that XCS deals with the adaptation of input query, demonstrating the usability for solving the problem whose detecting target is continuously updated. The very notable results of the experiments are that the accuracy of classifiers increases although the entire number of rule decreases in target training step (40-41) as shown in Table 5.

One of the main conclusion of this investigation is that we empirically show that the performance of XCS is dominantly affected by updating the rule, not the guarantee the noticeable rules from genetic algorithms. Of course, not only the genetic algorithm but also exploration which sometimes occurs help to create the proper rules but are just exploration processes that search the rule space, not for generating the notable classifier.

Also, in the process of training step, XCS learns and generates the rules and update the classifiers at once, decreasing the number of rules. This characteristic of training is shown in exploration-optimization (increase-decrease pattern of the number of rules) pattern of the number of rules in every scenario.

In the future works, we will study and analyze the roles of each element in training steps of XCS to propose the idea of developing the XCS/LCS algorithms for practical problems. The saturation of rule of XCS is fundamental problem of XCS (also of LCS) but recently are solvable using Self-Organized Classifiers [19] or XCS whose the rules are more complex [4, 10].

We simply applied the fundamental model of XCS. Therefore, it is possible to apply other alternatives or updated versions of XCS, for example, the hybrid model with another algorithm or approach, such as XCSR (XCS with autoencoder) [13], and XCSAM (XCS with adaptive action map) [14]. Then we could analyze the differences of inter-model to find the model specialized in target problem.

ACKNOWLEDGMENTS

This work was supported by Defense Acquisition Program Administration and Agency for Defense Developmentnder the contract. (UD160066BD)

REFERENCES

[1] Ajayi Adebowale, SA Idowu, and Otusile Oluwabukola. 2013. An overview of database centred intrusion detection systems. *Int. J. Eng. Adv. Technol* 3, 2 (2013), 273–275.
[2] Daniel Barbará, Rajni Goel, and Sushil Jajodia. 2003. Mining malicious corruption of data with hidden Markov models. In *Research Directions in Data and Applications Security*. Springer, 175–189.
[3] Salah Eddine Benaicha, Lalia Saoudi, Salah Eddine Bouhouita Guermeche, and Ouarda Lounis. 2014. Intrusion detection system using genetic algorithm. In *Science and Information Conference (SAI), 2014*. IEEE, 564–568.
[4] Martin V Butz, Pier Luca Lanzi, and Stewart W Wilson. 2008. Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Transactions on Evolutionary Computation* 12, 3 (2008), 355–376.
[5] Transaction Processing Performance Council. 2010. TPC BENCHMARK^{TM} E. http://www.tpc.org/tpce/.
[6] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security* 28, 1-2 (2009), 18–28.
[7] John H Holland and Judith S Reitman. 1978. Cognitive systems based on adaptive algorithms. In *Pattern-directed Inference Systems*. Elsevier, 313–329.
[8] Weiming Hu, Jun Gao, Yanguo Wang, Ou Wu, and Stephen Maybank. 2014. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics* 44, 1 (2014), 66–82.
[9] Yi Hu and Brajendra Panda. 2004. A data mining approach for database intrusion detection. In *Proc. of the Symposium on Applied computing*. ACM, 711–716.
[10] Muhammad Iqbal, Will N Browne, and Mengjie Zhang. 2017. Extending xcs with cyclic graphs for scalability on complex boolean problems. *Evolutionary Computation* 25, 2 (2017), 173–204.
[11] Ashish Kamra, Elisa Bertino, and Guy Lebanon. 2008. Mechanisms for database intrusion detection and response. In *Proc. of the 2nd SIGMOD PhD Workshop on Innovative Database Research*. ACM, 31–36.
[12] Wei Li. 2004. Using genetic algorithm for network intrusion detection. *Proc. of the United States Department of Energy Cyber Security Group* 1 (2004), 1–8.
[13] Kazuma Matsumoto, Yusuke Tajima, Rei Saito, Masaya Nakata, Hiroyuki Sato, Tim Kovacs, and Keiki Takadama. 2016. Learning classifier system with deep autoencoder. In *Evolutionary Computation (CEC), Congress on*. IEEE, 4739–4746.
[14] Masaya Nakata, Pier Luca Lanzi, and Keiki Takadama. 2015. Rule reduction by selection strategy in XCS with adaptive action map. *Evolutionary Intelligence* 8, 2-3 (2015), 71–87.
[15] Frank S Rietta. 2006. Application layer intrusion detection for SQL injection. In *Proc. of the 44th annual Southeast Regional Conf.* ACM, 531–536.
[16] Charissa Ann Ronao and Sung-Bae Cho. 2015. Mining SQL queries to detect anomalous database access using random forest and PCA. In *Int. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 151–160.
[17] Abhinav Srivastava, Shamik Sural, and Arun K Majumdar. 2006. Database intrusion detection using weighted sequence mining. *Journal of Computers* 1, 4 (2006), 8–17.
[18] Gary Stein, Bing Chen, Annie S Wu, and Kien A Hua. 2005. Decision tree classifier for network intrusion detection with GA-based feature selection. In *Proc. of Southeast Regional Conf.-Volume 2*. ACM, 136–141.
[19] Danilo V Vargas, Hirotaka Takano, and Junichi Murata. 2013. Self organizing classifiers and niched fitness. In *Proc. of Conf. on Genetic and Evolutionary Computation*. ACM, 1109–1116.
[20] Stewart W Wilson. 1995. Classifier fitness based on accuracy. *Evolutionary Computation* 3, 2 (1995), 149–175.