Expert Systems with Applications 36 (2009) 12036–12043

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An adaptive genetic-based signature learning system for intrusion detection

Kamran Shafi *, Hussein A. Abbass

School of Information Technology and Electrical Engineering, University of New South Wales @ Australian Defence Force Academy (UNSW@ADFA), Canberra, Australia

ARTICLE INFO

Keywords:
Rule-based systems
Knowledge extraction
Intrusion detection
Genetic-based learning
Incremental learning
Learning classifier systems

ABSTRACT

Rule-based intrusion detection systems generally rely on hand crafted signatures developed by domain experts. This could lead to a delay in updating the signature bases and potentially compromising the security of protected systems. In this paper, we present a biologically-inspired computational approach to dynamically and adaptively learn signatures for network intrusion detection using a supervised learning classifier system. The classifier is an online and incremental parallel production rule-based system.

A signature extraction system is developed that adaptively extracts signatures to the knowledge base as they are discovered by the classifier. The signature extraction algorithm is augmented by introducing new generalisation operators that minimise overlap and conflict between signatures. Mechanisms are provided to adapt main algorithm parameters to deal with online noisy and imbalanced class data. Our approach is hybrid in that signatures for both intrusive and normal behaviours are learnt.

The performance of the developed systems is evaluated with a publicly available intrusion detection dataset and results are presented that show the effectiveness of the proposed system.

1. Introduction

The nature of attacks and threats to computer systems have been co-evolving with the advances in computer security technology. Often, not long after the defences of information systems are strengthened that the computer criminals and other mis-adventurers find new ways of penetrating these systems. Thus adaptability, or the ability to learn and react to a consistently changing threat environment, is a key requirement for modern intrusion detection systems.

The two fundamental principles used in intrusion detection are the anomaly and misuse detection. The anomaly based approaches focus on modelling the normal behaviour of the protected system and flag deviations from this model as potentially harmful. While these techniques are aimed at detecting previously unknown attacks, there are a few that are implemented in the real world security systems. This is either because they are too sensitive and thus produce myriad of false alarms burying true alerts under them or that they require intensive processing cost rendering them infeasible for many real time implementations.

The misuse based approaches look for patterns of known attacks and vulnerabilities in the activities of protected systems. These techniques are also popularly referred to as signature based, because they generally rely on rules, or so called signatures, of known intrusions and vulnerabilities. Note, however, that rule-based profiling of normal behaviour (for anomaly detection) is also possible but is a relatively less explored area, especially in the signature self-learning class (Axelsson, 2000). The focus of this paper is this area.

Signature based systems are considered efficient and are used in most real time systems as they need less processing resources and produce fewer false alarms. The rules however need to be updated persistently in order to cope with the ever changing attacks and threats to the information system. Currently, rule creation and update is mainly a manual process (Bro, Snort). Thus signature based models suffer from a delay between the updates of signatures and the new attacks. Very often by the time signatures are made available, the damage had already been done. Since these systems rely on a manual creation of signatures, achieving adaptability is a harder problem for misuse detection systems. The anomaly detection systems by definition look for novel events; nonetheless they also need to adapt their learnt models of normal behaviour relative to changes in the environment.

A third hybrid approach is to combine the strengths of both misuse and anomaly detection in a single framework. Despite the fact that many researchers believe in the effectiveness of hybrid approaches (Axelsson, 2000; Kemmerer & Vigna, 2002; Bace & Mell, 2001), most practical intrusion detection systems remain signature based and a handful with anomaly based components (Vigna & Kemmerer, 1999).

In this paper, we propose a framework for dynamically and adaptively learning signatures using a biologically-inspired machine learning approach. The systems found in Nature, often referred to as complex adaptive systems, are highly robust and

* Corresponding author. Tel.: +61 418249510.
E-mail addresses: k.shafi@adfa.edu.au (K. Shafi), h.abbass@adfa.edu.au (H.A. Abbass).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.03.036

resilient that can adapt to environmental changes and constantly evolve their states for their betterment. The study of adaptation in natural systems is the basic theme of complex adaptive systems theory. Natural systems achieve adaptation in various ways such as by promoting biological diversity. Computational complex adaptive system based approaches mimic these natural models to achieve similar objectives in man-made systems. Several biologically-inspired techniques have been applied to the intrusion detection problem; some of the recent examples include (de Sá Silva, dos Santos, Mancilha, da Silva, & Montes, 2008; Lai, Chen, Jeng, & Chao, 2008).

Our approach to learning signatures for intrusion detection, therefore, is based on a genetic-based machine learning (GBML) technique called Learning Classifier Systems (LCS) (Holland et al., 2000). LCS are parallel production systems that have been designed to exploit the implicit parallelism of genetic algorithms (GA) (Goldberg, 1989). Genetic algorithms are implicitly parallel meta-heuristic search procedures that are inspired by the natural evolution process. The strength of LCS lies in their ability to adapt to the changes in the environment. There are three key features which make LCS highly desirable from intrusion detection viewpoint.

- LCS are rule-based learners that employ genetic algorithms as their generalisation mechanism to learn accurate, maximally general and interpretable rules. The rule-based representation lets domain experts to understand the learnt knowledge and the rules can easily be ported to existing signature based intrusion detection systems.
- LCS learn incrementally (i.e., they update their knowledge after seeing each input instance) but without needing to re-evaluate the whole model or to keep data instances in memory, a traditional approach to incremental learning. Thus they suit the streaming data requirements where the data is considered virtually lost after seeing it once.
- The flexible framework of LCS allows incorporating various representations and learning schemes with ease.

Our approach is hybrid in that the we learn rules for both intrusive and normal events. The rules are learnt dynamically (i.e., using machine intelligence and without the requirement of a domain expert), and adaptively (i.e., as the data arrives without needing to reevaluate the learnt model after seeing each data instance). We evaluate the base learning and signature extraction systems developed in this paper using a publicly available benchmark intrusion detection dataset, aka 1999 KDD Cup dataset.

The rest of the paper is organised as follows: In Section 2 background information to LCS, and other related concepts is provided. In Section 3 the signature extraction algorithm is introduced and explained in detail with a clear pseudo-code presentation. In Section 4 we discuss the modifications done in the underlying LCS. Section 5 presents our case study on the 1999 KDD Cup dataset. Experimental setup is described in detail including introduction to the dataset and results are presented and discussed. A comparison of performance, ruleset statistics and processing time between the base learning system, and the signature extraction system is presented. The paper is concluded in Section 6.

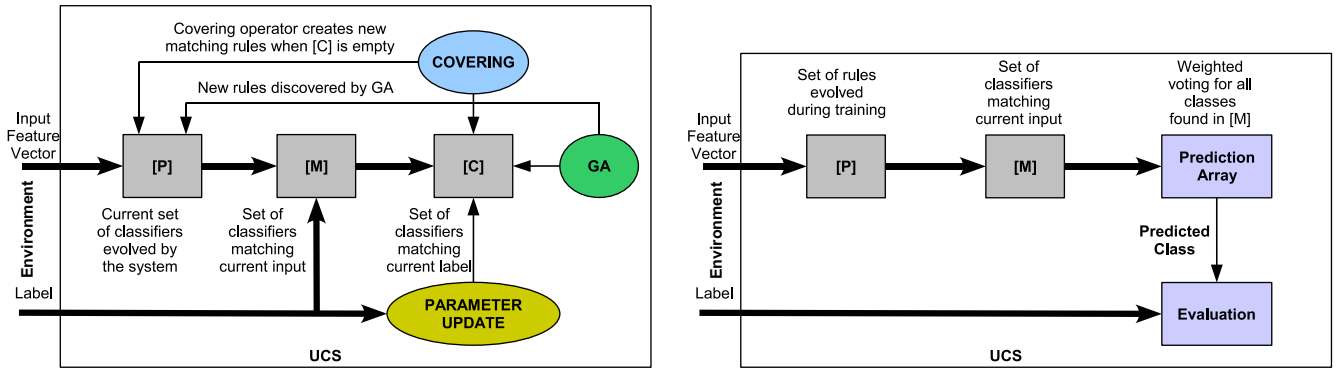
2. Learning classifier systems

LCS are rule-based systems that combine a sequential covering algorithm with a Genetic Algorithm (GA) based search technique to learn rules in a disjunctive normal form. GA are meta-heuristic search techniques that loosely imitate the process of natural evolution by simulating principles of natural selection and reproduction. Similar to natural systems, which evolve over many generations to adapt to their environments using natural selection, reproduction and diversification, GA evolve a set of candidate solutions to adapt to the requirements of a problem.

Unlike traditional rule induction approaches (specific-to-general or general-to-specific search), GA search large hypothesis spaces by first randomly generating a collection of hypotheses (i.e., individuals), referred to as *population*. It then iteratively selects the best individuals from this population and reproduce the next generation of individuals by crossing over and mutating the individuals in the selected subset. The selection of best individuals is performed stochastically according to some desired performance measure, denoted as the *fitness* of the individual.

The GA module empowers LCS to adaptively learn new rules as well as provide generalisation mechanism that enables the system to classify future cases. Traditional machine learning paradigms, such as reinforcement or supervised learning, are incorporated to evaluate the *fitness* or quality of rules. The rules in LCS are referred to as *classifiers*, thus the name Classifier Systems.

There are two common approaches to LCS. Pittsburgh style LCS represent complete rule sets as an individual classifier. Michigan style LCS on the other hand consider each individual rule as a classifier. It is the latter approach which we are concerned with in this paper. Specifically, we will focus our work on a supervised learning classifier system (UCS) (Bernadó-Mansilla & Garrell, 2003). UCS evolves a population $[P]$ of classifiers where each classifier consists of a rule, which includes a condition (essentially a conjunction of predicates) and an action or the predicted class, and a set of parameters. The two main parameters are *accuracy* (*acc*), which measures the correctness of a rule and *fitness* (*F*). Accuracy is calculated as a



(a) Training Phase (b) Test Phase

Fig. 1. A typical UCS working cycle.

ratio between the number of times a classifier correctly classified an example and its *experience* (i.e., the number of times it has been fired). The fitness is then updated as a direct function of accuracy.

Fig. 1 depicts the interaction of different components and a typical working cycle of UCS in both training and test phases.

During training UCS learns from the labelled examples incrementally in a supervised mode. In each iteration an instance is presented to the system and a *matchset* [*M*] is built of all the matching classifiers in [*P*]. A classifier matches an instance only if all of its conditions satisfy the corresponding feature values of the input instance. The parameters of all classifiers participating in the [*M*] are then updated. Those classifiers in [*M*] that predict the same class as the label of the current input example form the *correctset* [*C*]. If [*C*] is empty then the *covering* operator is used to create a new matching rule with the same class as the label of the matched example. Finally GA is applied to the [*C*], during which two parents from [*C*] are selected stochastically, proportional to their fitness. Two offspring are generated by reproducing, crossing-over and mutating the parents. The resultant offspring are first checked to see if they can be subsumed by their parents (i.e., if the parents are more general than children in all attributes, sufficiently experienced and accurate) then the weight of the parents (kept by a parameter called *numerosity*) is incremented and children are discarded. Otherwise children are inserted in the population. If the insertion of the new classifier causes the population size to exceed its user defined maximum threshold then a classifier is deleted stochastically to create room for the new one.

During the test phase learning and induction processes do not take place and the system predicts the class of test cases using the population of classifiers it has evolved during training. For each test input a [*M*] is formed as usual and a *system prediction* for each class is calculated as a fitness-weighted vote. The class with the highest prediction vote is selected as the output.

3. A real time signature extraction algorithm

A key feature of LCS is the interpretability of its learned knowledge in the form of simple rules. However, evolutionary search used in LCS can produce many overlapping generalisations and may not converge to a compact representation (Wilson, 2001). A large number of rules can in turn reduce the readability rendering it difficult to comprehend by the human experts. In addition, the processing time increases exponentially with rise in the number of rules, which is a major issue for time critical applications like intrusion detection.

One way this problem is addressed in the literature is by post-pruning the evolved rule sets at the completion of training runs (Dixon, Corne, & Oates, 2003; Wilson, 2001). The concentration of these algorithms is to find a subset of the post training rule population that performs equivalent to the actual population on the training set. Many recent data mining applications including network intrusion detection, however, require dealing with data as it arrives. These data streams are often time varying in nature and a data instance once processed is considered virtually lost because the unprecedented amount of data makes it impractical to either fully store or visit it multiple times. It is thus imperative to devise data mining techniques to deal with such data on the fly and adaptively.

In this section, we present an algorithm to extract accurate and maximally general classifiers, which we refer to as *signatures*, in real time (i.e., as soon as they are discovered by UCS). We will refer to this system as *UCSSE* for *UCS with real time Signature Extraction* system.

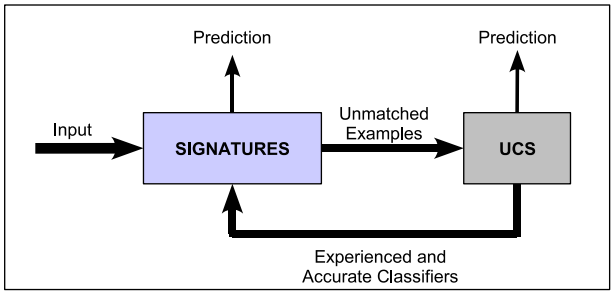


Fig. 2. UCS with Real time *Signature* Extraction System (UCSSE).

3.1. Overview

Fig. 2 shows a block diagram of the proposed signature extraction system.

The *signature set* [*S*] (or the knowledge base) consists of the optimal classifiers extracted during the operation of UCS. An input from the environment is first presented to [*S*] whereby [*M*] is generated using the current input label and the accuracy of those signatures participating in [*M*] are updated similar to UCS. The discovery component of UCS (i.e., the GA) is bypassed when the system is run through [*S*]. An input is escalated to UCS only if no matching signature could be found in [*S*], in which case standard UCS takes over and runs its performance and discovery components using [*P*] for a certain number of trials. Meanwhile, the extraction process of accurate and experienced classifiers from [*P*] to [*S*] is triggered periodically.

Initially, [*S*] is empty and the system runs mainly through [*P*] getting enough exploration opportunities. The operation is shifted gradually to [*S*] as it starts getting populated. The transition completes when the system discovers the best map of the input space. The control is then transferred to [*S*], in which case the evolutionary search is completely halted and the system is made to run from [*S*]. A pruning step in [*S*] is carried out when the average experience of the signatures in [*S*] reaches a threshold. In the pruning step, the signatures that are inaccurate and have a below average experience are deleted from [*S*]. If the deletion causes a covering gap, control is handed back to [*P*] and the process is repeated until the system stabilizes to run from [*S*] at which point the learning process can be stopped. The complete pseudo-code for the algorithm is given in the next section.

To understand the significance of the proposed system, imagine an always *on* intrusion detection system which identify illegitimate activities in a production network. The signatures learned from UCS can work as a first line of defence in such an environment. As new intrusions emerge the system could adapt by learning new signatures using the main UCS application. In a steady state condition one would expect most of inputs being covered by signatures and the genetic search of UCS only being invoked when there is an indication of the concept change in the environment.

3.2. Pseudo-code

A detailed pseudo-code of the algorithm is given below.

3.2.1. RUN EXPERIMENT

This is the main routine and works as discussed above. T_{opt} corresponds to the minimum number of time steps for which control is switched to UCS when a [*M*] through [*S*] is found empty. Exploit or test phase works similarly except that there is no extraction or UCS learning during this phase and a class with the highest vote calculated based on the accumulated accuracy of all matching signatures is predicted. The vote is calculated solely on the aggregative accuracy of the matching signatures.

Procedure 1. RUN EXPERIMENT

```
1:  for each input σ do
2:    [S] ← EXTRACT SIGNATURES out of [P]
3:    if [S] ≠ ∅ then
4:      [M] ← GENERATE MATCH SET out of [S] using σ
5:      if [M] ≠ ∅
6:        act ← SELECT ACTION according to σ
7:        UPDATE SET [M] according to act
8:      else
9:        switch to standard UCS for next T_opt time steps
10:     endif
11:   else
12:     run a standard UCS trial
13:   endif
14: endfor
```

3.2.2. EXTRACT SIGNATURES

The extraction from $[P]$ to $[S]$ is triggered after a sufficient number of trials since last extraction has occurred (T_{ext}) and there has been at least one standard explore cycle ($N_{[P]}$) during this time. This ensures that the population is not scanned unnecessarily and the extraction step takes place only when the population contains at least some newly discovered optimal classifiers.

In addition, a pruning procedure in $[S]$ is invoked if the average set experience exceeds a multiple (controlled by C) of θ_{xexp}.

Procedure 2. EXTRACT SIGNATURES ($[P]$)

```
1: if N_[P] ⩾ 1 ∧ T_ext ⩾ T_opt then
2:   for each classifier cl in [P] do
3:     if cl IS QUALIFIED FOR INSERTION then
4:       INSERT cl in SIGNATURE SET
5:     end if
6:   end for
7: end if
8: if [S].experience ⩾ C.θ_xexp
9:   PRUNE SET [S]
10: end if
```

3.2.3. IS QUALIFIED FOR INSERTION

Any classifier with experience greater than θ_{xexp} and accuracy greater than θ_{xacc} is considered qualified for insertion in $[S]$ as a *signature*. The values of extraction thresholds (i.e., θ_{xexp} and θ_{xacc}) are necessarily problem dependent. Furthermore, in the presence of noise or imbalanced class distribution in the training data, the accuracy and experience of optimal rules in UCS would vary according to the levels of noise and imbalance. Unfortunately, it is often not possible to a priori determine these complexities and subsequently the best possible accuracy that a classification algorithm can achieve in most real world problems. Consequently, both of these parameters in UCSSE are adapted during the learning process.

A simple procedure to adapt θ_{xexp} is given in Procedure 3. At each extraction step the update procedure looks for the most accurate rule in $[P]$ that has enough experience ($> \theta_{xexp}$) to be extracted as a signature. If this accuracy value (referred to as *Supply Accuracy*) is higher than the average accuracy in $[S]$ then θ_{xacc} is adjusted to the new value. Notice that the accuracy for each class is computed independently. This allows to handle varying levels of noise in different classes.

Procedure 3. UPDATE EXTRACTION ACCURACY

```
1: θ_xacc ← ACC0
2: for each Class c do
3:   Δ = GET SUPPLY ACCURACY(c)
4:   if Δ > [S]_c.accuracy then
5:     θ_xacc[c] = Δ
6:   end if
7:   end for
```

Similarly θ_{xexp} is adapted according to Eq. (1), where C corresponds to the number of classes and I_a to the total number of instances belonging to the class of the current example calculated over a window to cater for the streaming data.

$$\theta_{xexp_t} = C \cdot \theta_{xexp} \cdot \frac{I_a}{\sum_{i=1}^{C} I_i}. \tag{1}$$

3.2.4. INSERT IN SIGNATURE SET

Two new generalisation operators (i.e., Disjoin and Merge) are introduced in this procedure that aim to minimise overlap and conflict between signatures. In addition, a signature is checked for either way subsumption when inserting in $[S]$ for a higher generalisation pressure. Pseudo-code for the insertion procedure is given in Procedure 4.

Procedure 4. INSERT IN SIGNATURE SET (qcl)

```
1:  subsumed = false
2:  for each cl in [S] do
3:    if qcl.class ≠ cl.class then
4:      if DOES OVERLAP (qcl, cl) then
5:        DISJOIN (qcl, cl)
6:      end if
7:    end if
8:  end for
9:  for each cl in [S] do
10:   if qcl.class == cl.class then
11:     if cl SUBSUMES qcl then
12:       subsumed = true
13:       break
14:     else if qcl SUBSUMES cl then
15:       DELETE cl
16:       continue
17:     else if DOES OVERLAP (qcl, cl) then
18:       subsumed = MERGE (qcl, cl)
19:       if subsumed then
20:         break
21:       end if
22:     end if
23:   end if
24: end for
25: if subsumed = false then
26:   ADD qcl TO [S]
27: end if
```

Every time a new signature is found and inserted in $[S]$, it is first scanned against each signature of the opposite class in $[S]$. If an overlap is found between any two signatures, the one with lower accuracy and lower experience is *contracted* to resolve the overlap. Next the signature being inserted is scanned against each existing signature with the same class prediction. If an overlap is found that exceeds a threshold θ_{ol} and cannot be resolved by traditional

subsumption operator, one of the signatures is expanded to subsume the other. For brevity we do not provide the pseudo-codes for DISJOIN, MERGE and DOES OVERLAP procedures.

3.2.5. PRUNE SET

Since the algorithm is tuned to extract optimal classifiers as soon as they are evolved by UCS, it is likely that the rules which meet the extraction criteria but are actually slightly overgeneral or less than maximally general are extracted to $[S]$. In this case $[S]$ can contain many rules which over time become inaccurate as they start matching more instances of other classes or the rules which are less experienced as they match fewer instances than the maximally general classifiers. To overcome this issue we can either constrict the extraction criteria by increasing extraction thresholds or introduce a periodic pruning routine which cleans $[S]$ from such classifiers. Since the former approach can unnecessarily delay the retrieval of optimal rules we adopt the later approach of greedy extraction and periodic pruning in $[S]$.

Procedure 5. PRUNE SET ($[S]$)

```
1: for each classifier cl in [S] do
2:   if cl.accuracy < θ_dacc then
3:     DELETE cl
4:   end if
5: end for
6: SORT [S] according to experience
7: for i = 1 to i < [S].size do
8:   prevExperience ← S[i − 1].experience
9:   currExperience ← S[i].experience
10:  if prevExperience/currExperience < θ_dexp then
11:    cutoff ← i
12:  end if
13: end for
14: for i = cutoff to i < [S].size do
15:   DELETE classifier
16: end for
```

Pseudo-code for the pruning procedure is given in Procedure 5. During pruning first all classifiers which have accuracy below θ_{dacc} are deleted from $[S]$. Next, $[S]$ is sorted in descending order of classifier experience. A ratio between the experience of each consecutive pair is then calculated. If this ratio exceeds θ_{dexp}, all remaining signatures are deleted from $[S]$. Note that every time a new classifier is extracted to $[S]$ from $[P]$ the experience of all signatures is reset to zero (not shown in Procedure 2 for brevity). This ensures that all signatures are weighted equally when considering for deletion.

Clearly, the deletion accuracy threshold used for pruning inaccurate classifiers is also subjective, similar to the extraction thresholds, and can not be determined in advance for problems where the optimal classification accuracy is not known due to noise and other complexities. Consequently, θ_{dacc} is adapted according to Eq. (2), where Δ varies exponential linearly between $[0,\overline{[S].accuracy}]$ based on the *error* signal between the desired $[S]$ based trials and the current feedback (i.e., the number of $[S]$ based trials since the last deletion step).

$$\theta_{dacc}(c) = \overline{[S]_c.accuracy} - \Delta_c. \tag{2}$$

4. UCS modifications

Initial investigations with the original UCS (Shafi, Abbass, & Zhu, 2009) on KDD Cup dataset showed that it achieves low accuracy on almost all classes available in the KDD Cup dataset, generates high number of false positives and achieves high cost per example score. Subsequently, we introduced several modifications to the original UCS listed below:

- Distance metric – In UCS, if a test case is not covered by the evolved population then a class is predicted randomly. We introduced a distance metric based function that uses the existing rule set to find the nearest match to an uncovered test case rather than choosing randomly.
- Biased accuracy function – A new function is introduced to calculate rule accuracy based on the rate of misclassifications and frequency of input examples in each class. The new accuracy function referred to as class-distributive accuracy is calculated as:

$$acc = \frac{\frac{exp_p}{\sum_{i=1}^{C} exp_i} \cdot \frac{\sum_{i=1}^{C} I_i}{I_p}}{\sum_{i=1}^{C} \left[\frac{exp_i}{\sum_{j=1}^{C} exp_j} \cdot \frac{\sum_{j=1}^{C} I_j}{I_i} \right]} \tag{3}$$

  where C is the number of classes, exp_p corresponds to the experience of a classifier in the class it is predicting and I_p is the frequency of instances of the predicted class received so far by the system. The idea behind the class-distributive accuracy is to give more weight to the minority class rules. Since, the number of examples are calculated online, the parameter estimation reaches near equilibrium as more data arrives to the system.
- GA rate adaptation – In UCS GA provides the generalisation mechanism and is responsible for discovering new rules. GA is applied to the correctsets ($[C]$) and hence the reproduction opportunities given to each class (or so called niches) depend on the level of imbalance in the data. Since GA rate is a user-defined parameter in UCS, a new function is introduced to adapt the rate of GA application online. It is calculated as

$$\theta_{ga_t} = C \cdot \theta_{ga} \cdot \frac{\sum_{cl_i \in [C]} exp_a}{\sum_{cl_i \in [C]} exp}, \tag{4}$$

  where exp_a corresponds to the experience of a classifier in the class of the current example and exp is the sum of experience in all classes.
- Fitness sharing – Fitness sharing is a commonly used technique in GA for promoting diversity in a population and to effectively search multi-modal search spaces. In LCS fitness sharing is applied to alleviate the problem of overgeneral rules that can become a bottleneck in imbalanced class problems. In UCS fitness sharing has recently been introduced (Orriols-Puig & Bernadó-Mansilla, 2006) and thus we incorporate it in our modifications.

The UCS with modifications will be referred to as UCSm in the rest of the paper and its performance will be compared with the original UCS in the next section.

5. Experiments

In this section we will evaluate UCS, UCSm and UCSSE on a publicly available intrusion detection dataset using standard measures used in this domain.

5.1. KDD Cup dataset

The KDD Cup intrusion detection dataset is publicly available from (Hettich & Bay, 1999) and provides an easy access and a common platform for the evaluation of intrusion detection systems. Separate training and test datasets were provided during the competition.

The training dataset consists of approximately 5 million connection records extracted from 7 weeks of network traffic dump. A 10% version of the training dataset is also provided which contains almost half a million record with similar class distribution as the original dataset. The test set contains approximately 0.3 million connection records extracted from an additional two weeks of traffic dump.

Each connection record in the dataset is a labelled feature vector consisting of 41 features in total including four discrete, three nominal and 34 continuous valued features. The labels correspond to either a normal connection or an attack. There is a total of 39 intrusion types including both training and test datasets. All of these types can be classified into four major attack categories (i.e., Probe, Denial Of Service (DOS), User to Root (U2R) and Remote to Local (R2L)). The dataset provides a challenging problem for machine learning systems and contains severe class imbalance. Please refer to Hettich and Bay (1999) for more details.

5.2. Evaluation metrics

The contestants of 1999 KDD Cup were evaluated on two measures. First, the classification accuracy for each class was reported by using the standard confusion matrix approach. A confusion matrix is a $n \times n$ square matrix where n represents the number of classes, which summarises the errors made by a classification model. It is calculated by comparing the actual class labels in the test set with that of the predicted ones by a classifier. Second, a cost per example (CPE) score was assigned to each classification model using a cost matrix based on the rarety of the classes. The CPE is computed by multiplying the cost and confusion matrices and dividing the sum of the resultant matrix by total number of test instances. A lower cost-based score means better classification model. In our results, we will also use these two measures to report the accuracy of our classification models.

In addition, *false alarm rate* and *hit rate* are two other measures that are used commonly to measure the performance of intrusion detection systems (IDSs). False alarms refer to the benign activities reported as intrusions by an IDS whereas hits refer to the true intrusions flagged by an IDS regardless of their particular type. Table 1 illustrates the calculation of these two measures along with the overall accuracy using confusion matrix.

5.3. Methodology

In the experiments, UCS is trained using the 10% KDD training dataset and then the learned model (i.e., the rule set) at the end of each training pass is used to classify test dataset. As mentioned, 10% dataset consists of half a million records and maintains almost the same class distribution as the full dataset. It is important to note that UCS is trained using only a single pass through the data, unlike the traditional batch learning systems. Although multiple passes can be made through the training data; in this paper, our focus is to learn rules in real time. Since UCS is a stochastic algorithm, each experiment is repeated 30 times with a different seed and results are averaged over all runs. In our UCS implementation, each feature type (i.e., real, integer, binary and nominal) in the KDD dataset is handled independently according to its respective representation as discussed in Section 2. The nominal attributes are coded using a sparse representation such that each category in the nominal attribute is represented by a binary digit.

To ensure a fair comparison between UCS and UCSSE we also implemented Dixon et al. (2003) rule reduction algorithm to clean the post training rule populations evolved by UCS. Dixon's algorithm has shown to be magnitude faster than Wilson's rule reduction algorithm (Wilson, 2001) while achieving equivalent performance in terms of test accuracy. The actual algorithm is proposed and tested for XCS but we adopted it easily for UCS.

There are quite a few UCS parameters that need external setting. To ensure the optimal performance we used the recommended parameter settings (Butz, 2004) to suit the KDD dataset. For UCSSE we set $T_{opt} = 5000$, $C = 10$ and $\theta_{ol} = 0.4$. Dixon's algorithm require two parameters to prune *unqualified* rules, viz., θ_{exp} and θ_{acc}, which are set externally. For a fair comparison with the adaptive parameter tuning of UCSSE we set these two parameters to the average experience and accuracy values of post-training rule sets.

5.4. Results and discussion

First, we compare the performance of the original and modified UCS systems on KDD Cup dataset given in Tables 2 and 3. The numbers printed in bold are significantly better than the other system based on a pair-wise ttest at 99% confidence level, while the one in italics are significantly worse at the same level.

Table 2 shows per class test accuracy achieved by the two systems along with their confusion matrices. As can be seen UCSm achieves significantly better accuracy on all classes except R2L where its accuracy is not significantly different than UCS.

Overall modified UCS improves the baseline accuracy by almost 10% and achieve an overall accuracy of 92.03% (Table 3). UCSm also outperforms UCS on all other measures as it achieves significantly lower false alarm rate and CPE score and significantly higher hit

Table 1
Confusion matrix for a two-class problem and evaluation metrics.

	Actual	Predicted	
		Normal	Attack
accuracy $= \frac{a+d}{a+b+c+d}$	Normal	a	b
false alarm rate $= \frac{b}{a+b}$	Attack	c	d
hit rate $= \frac{d}{c+d}$			

Table 2
Comparison of confusion matrices between UCS and UCSm. The numbers in **bold** are significantly better at 99% confidence level using a pairwise ttest.

		Predicted					Accuracy
		Normal	Probe	DOS	U2R	R2L	
UCS							
Actual	Normal	59062	435	396	356	343	97.47 (0.36%)
	Probe	419	2548	391	432	376	61.17 (9.22%)
	DOS	12282	7274	195674	7271	7352	85.13 (7.89%)
	U2R	34	7	8	10	10	15.10 (4.87%)
	R2L	14920	375	279	267	507	3.10 (1.58%)
UCSm							
Actual	Normal	60215	181	118	48	32	**99.38 (0.07%)**
	Probe	758	3140	254	13	3	**75.36 (0.76%)**
	DOS	6928	438	222423	6	58	**96.77 (0.14%)**
	U2R	49	1	0	15	5	**21.52 (4.19%)**
	R2L	15649	205	15	19	458	2.80 (1.67%)

Table 3
Comparison of other key performance parameters between UCS and UCSm.

Measure	UCS	UCSm
Other performance measures		
Overall accuracy	82.89 (5.85)%	**92.03 (0.14)**%
FA rate	2.53 (0.36)%	**0.62 (0.07)**%
Hit rate	88.96 (2.05)%	**90.66 (0.16)**%
CPE	0.41 (0.10)	**0.26 (0.00)**

Table 4
Comparison of number of rules evolved by UCS and UCSm.

Class	UCS	UCSm
Number of rules		
Normal	6068(64)	**5384(33)**
Probe	510(24)	*527(17)*
DOS	916(53)	*1246(27)*
U2R	105(8)	*204(11)*
R2L	180(13)	*259(14)*
Overall	7779(14)	**7620(21)**

Table 5
Comparison of number of rules, test set accuracy and other performance measures between UCS, UCSD and UCSSE on the KDD Cup dataset.

Class	UCS	UCSD	UCSSE
Number of Rules			
Normal	5384(33)	386(12)▲	272(54)◆
Probe	527(17)	43(2)◆	129(8)▲
DOS	1246(27)	61(3)▲	60(15)▲
U2R	204(11)	4(1)◆	24(2)▲
R2L	259(14)	13(2)◆	25(14)▲
Overall	7620(21)	508(12)▲	510(63)▲
Test accuracy (%)			
Normal	99.38(0.07)	98.63(0.36)◊	99.16(0.26)△
Probe	75.36(0.74)	75.81(1.42)	75.54(0.81)
DOS	96.77(0.14)	96.95(0.06)◆	96.73(0.06)
U2R	21.52(4.12)	7.43(5.04)◊	21.33(4.96)
R2L	2.80(1.64)	2.55(1.84)	2.59(1.94)
Overall	92.00(0.00)	92.00(0.00)	92.00(0.00)
Other performance measures			
Measure			
FA Rate (%)	0.62(0.07)	1.37(0.37)◊	0.84(0.26)
Hit Rate (%)	90.66(0.16)	90.72(0.15)	90.68(0.16)
CPE	0.26(0.00)	0.26(0.00)	0.26(0.00)

rate than UCS. Also note that the modified UCS produces much more stable outcome than the baseline system.

Although the modified UCS achieves significantly better accuracy on all classes and overall than the baseline UCS, the accuracy on two rare classes especially R2L is quite low. The two most rare classes (i.e., U2R and R2L) have a very poor representation in the training data (an imbalance ratio of almost 1/7500 for U2R and 1/347 for R2L with respect to the majority DOS class). Under such sparse representation this becomes a very hard generalisation problem for any machine learning system. In fact, the winner of the KDD Cup achieved 13.2% and 8.4% accuracy on these two classes, respectively. In real world scenarios however, the percentage of intrusive activities is much higher than given in the KDD Cup or DARPA datasets (McHugh, 2000). Nonetheless, UCSm shows competitive results despite being an online (or single pass) and incremental rule-based learner. All of these characteristics are very important from the real time intrusion detection viewpoint.

Table 4 shows the average number of rules evolved by the two systems at the end of training runs. Although UCSm evolves significantly less number of rules for Normal class and hence overall, the difference is only marginal. However note that the number of rules are better balanced class-wise in UCSm than UCS. The above results suggest that high number of rules remain an issue despite the improvement in the overall performance of the system.

Table 5 provides a comparison of three systems (i.e., UCS[1], UCS with Dixon's rule reduction algorithm, referred to as UCSD, and UCSSE). Notice that UCSm is used as the base learner in both signature extraction system and UCSD. The results are averaged over 30 runs. The statistical significance is tested using a pairwise ttest at a significance level of 99%. A ▲ is used if UCSD or UCSSE is significantly better (higher in test accuracy and hit rate, and lower in number of rules, false alarm rate and CPE score) than UCS. A ◆ shows that UCSD or UCSSE is significantly better than both UCS and the other system. Similarly a △ denotes that UCSD or UCSSE is significantly worse (lower in test accuracy and hit rate, and higher in number of rules, false alarm rate and CPE score) than UCS. A ◊ denotes that UCSD or UCSSE is worse than the other two systems.

The top tabular shows the comparison of post training rule statistics in the three systems. First note that UCSSE retrieves almost same number of signatures as obtained by UCSD and thus achieves almost 15 times reduction on average in UCS rule set sizes. The biggest saving is achieved in the Normal class rules where UCSSE retrieves significantly less number of signatures than both other systems. As mentioned earlier, learning the signatures for both normal and intrusive activities allow providing both rule-based misuse and anomaly detection capability. Also note that the Dixon's rule reduction algorithm is applied after completely training UCS population and hence is expected to perform better than UCSSE. UCSSE despite being operating in real time performs competitively.

The middle tabular presents the per class test set accuracy achieved by the three systems. As shown there is no significant difference between the performance of these three systems as all of them achieve similar test accuracies overall. This is still a positive outcome since both UCSD and UCSSE achieve this with almost 15 times less number of rules than UCS. Also notice that UCSD performs poorly on U2R class in comparison to both other systems whereas UCSSE achieves similar accuracy on this class as UCS.

The bottom tabular presents the comparison of other performance measures achieved by the three systems. All systems perform at par on hit rate and CPE score metrics. Both UCSD and UCSSE however produce significantly higher numbers of false alarms than UCS, with UCSD producing the worst figures. This shows that a tradeoff exists between the size of the rule set and the false alarm rate.

Table 6 shows the average CPU time in minutes taken by the three systems during training and testing through the KDD Cup dataset. The experiments are run on a high performance parallel computing Linux Beowulf cluster with 152 dual 3 GHz Pentium-IV computing nodes. The time spent in training a system corresponds to a single pass through the training data while UCS works in an exploration mode, whereas the evaluation time corresponds to the exploitation phase during which each test instance are labelled using the evolved population or signatures. In addition to the training and evaluation time, time taken by UCSD to post process the UCS population is also reported.

UCSSE improves the margin on processing time between the three systems as it reduces the processing time by almost 5 and 3.5 times than UCS and UCSD respectively. This corresponds to almost 3421 instances processed per second in comparison to 700 and 938 instances per second processed by UCS and UCSD, respectively. The post processing step in UCSD adds an overhead on the training time of UCS, however, this is compensated during the evaluation where UCSD processes magnitude faster than UCS. In UCSSE, the savings in computational time is achieved with the help of smaller rule sets and avoiding unnecessary discovery cycles. As

Table 6
Comparison of CPU time used by UCS, UCSD and UCSSE on the KDD Cup dataset.

Time	UCS	UCSD	UCSSE
Training	71.54(2.46)	71.47(2.37)	13.36(0.68)◆
Post-processing	0.00(0.00)	17.62(0.56)◊	0.00(0.00)
Evaluation	52.58(2.90)	3.48(0.18)◆	12.02(3.03)▲
Total	124.12(4.84)	92.57(2.74)▲	25.38(2.98)◆

[1] For clarity we use UCS instead of UCSm hereon.

the coverage of signatures improve over time we could expect more savings in the computational time.

Note that each record in KDD Cup datasets corresponds to a traffic session between two computers which generally includes tens to hundreds of packets, amounting to thousands of bytes, exchanged between the communicating machines. For instance, 5 million training records in KDD Cup dataset were built from 7 weeks of raw traffic amounting to several gigabytes of raw traffic dump. Our estimation through experiments with real traffic collected at a university server shows that UCSSE can handle more than 10 megabits per second of traffic at the current speed but with a different set of features. Moreover our algorithms can benefit from better implementation techniques such as rule indexing and hashing in further improving the processing time for real time deployment.

The empirical evidence presented in this section demonstrates the usefulness of the signature extraction system for the intrusion detection problem. UCSSE with modified UCS setup is able to retrieve almost 15 times lesser number of rules in real time without a loss in overall accuracy. This allows almost five times saving in the processing time. We believe that the processing time would be improved further with larger datasets or in a streaming data environment and with more sophisticated signature matching algorithms.

6. Conclusion

A framework, named UCSSE, for real time extraction of maximally general rules (or signatures) from a supervised learning classifier system (UCS) is presented. The algorithm automatically and adaptively identify and extract signatures for normal and intrusive activities as they are being discovered by UCS. Two new generalisation operators (i.e., *merge* and *contract*) are introduced in UCSSE that modify rule boundaries to minimise overlap and conflict among signatures. Control mechanisms are provided for the online adaptation of critical UCSSE thresholds to deal with noise and imbalanced class data. We have also incorporated several modifications in UCS. The extended UCS shows a significant improvement in all performance metrics over baseline UCS on the KDD Cup dataset.

The utility of UCSSE is demonstrated on a publicly available intrusion detection dataset and its performance is compared with an offline post-training rule pruning algorithm (UCSD). Both UCSD and UCSSE achieve 15 times rule reduction in the original rule set size without any performance degradation overall. UCSD however produces significantly higher number of false alarms than both UCS and UCSSE. UCSSE evolves 510 signatures that include signatures for both intrusive and normal activities in a single pass through the data achieving a five times reduction in processing time of UCS.

In conclusion, UCSSE provides a framework for an automatic and adaptive discovery of signatures for intrusion detection using learning classifier systems as a base learner. The learning of both normal and intrusive activities allows UCSSE to provide hybrid detection capabilities. This work is a step towards real time intrusion detection, which require dealing with huge volumes of data on the fly. In future we intend to test UCSSE with traffic collected from the real networks and implement an interface between UCSSE and existing signature based intrusion detection systems such as Snort.

Acknowledgements

This work is funded by University College Postgraduate Research Scholarship (UCPRS). Most of these experiments were run on the Australian Center for Advanced Computing (AC3) super computing facilities.

References

Axelsson, S. (2000). *Intrusion detection systems: A survey and taxonomy*. Technical report 99-15, Chalmers University of Technology, Department of Computer Engineering, Göteborg, Sweden.

Bace, R., & Mell P. (2001). *Intrusion detection systems: A survey and taxonomy*. Technical Report 800-31, National Institute of Standards and Technology.

Bernadó-Mansilla, E., & Garrell, J. M. (2003). Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation, 11*(3), 209–238.

Bro. The open source network intrusion detection system. <http://www.bro-ids.org/>.

Butz, M. V. (2004). *Rule-based evolutionary online learning systems: Learning bounds, classification, and prediction*. PhD thesis, University of Illinois at Urbana-Champaign.

de Sá Silva, L., dos Santos, A. C. F., Mancilha, T. D., da Silva, J. D. S., & Montes, A. (2008). Detecting attack signatures in the real network traffic with ANNIDA. *Expert Systems with Applications, 34*(4), 2326–2333.

Dixon, P. W., Corne, D. W., & Oates, M. J. (2003). A ruleset reduction algorithm for the XCS learning classifier system. In *Learning classifier systems: Fifth international workshop, IWLCS 2002, Granada, Spain, September 7–8, 2002: Revised Papers* (pp. 20–29).

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addision-Wesley Publishing Company, Inc.

Hettich, S., & Bay, S. D. (1999). *The UCI KDD Archive*. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

Holland, J. H., Booker, L. B., Colombetti, M., Dorigo, M., Goldberg, David E., Forrest, S., et al. (2000). What is a learning classifier system. *Learning Classifier Systems: From Foundations to Applications, 1813*, 3–32.

Kemmerer, R. A., & Vigna, G. (2002). Intrusion detection: A brief history and overview. *Computer, 35*(4), 27–30.

Lai, G. H., Chen, C. M., Jeng, B. C., & Chao, W. (2008). Ant-based IP traceback. *Expert Systems with Applications, 34*(4), 3071–3080.

McHugh, J. (2000). Testing Intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information System Security, 3*(4), 262–294.

Orriols-Puig, A., & Bernadó-Mansilla, E. (2006). A further look at UCS classifier system. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation Workshop Program*, ACM Press, 2006.

Shafi, K., Abbass, H. A., & Zhu, W. (2009). Intrusion detection with evolutionary learning classifier systems. *Natural Computing, 81*(1), 3–27.

Snort. The open source network intrusion detection system. <http://www.snort.org/>.

Vigna, Giovanni., & Kemmerer, Richard A. (1999). NetSTAT: A network-based intrusion detection system. *Journal of Computer Security, 7*(1), 37–71.

Wilson, S. W. (2001). Compact rulesets from XCSI. In *Revised papers from the 4th international workshop on advances in learning classifier systems* (pp. 197–210).