How Many Parachutists will be Needed to Find a Needle in a Pastoral?

Akira Imada
Brest State Technical University
Moskowskaja 267 Brest 224017 Belarus
akira@bstu.by

***Abstract*— This article is a consideration on computer network intrusion detection using artificial neural networks, or whatever else using machine learning techniques. We assume an intrusion to a network is like a needle in a haystack not like a family of iris flower, and we consider how an attack can be detected by an intelligent way, if any.**

I. Introduction

The parachute drop went smoothly ... slithering down the chute and out into space ... Flick landed perfectly, with her knees bent and her arms tucked into her sides as she fell to the ground ... She folded her parachute into a neat bundle, then set out to find the other Jackdaws. – "Jackdaws" by Ken Follett.

Most banks nowadays facilitate their ATM (Automated Teller Machine) in which we may have a personal account to which we can access with PIN-code, usually four digits of decimal numeral. For security reason, if we failed to enter the PIN correctly more than three times in a row, the PIN would loose its validity thereafter. Then what we are curious is, "How many trials would be needed for random challenges to reveal the secret PIN if an infinite number of trials were permitted?" Let's formalize this problem.

Problem 1 (Breaking a PIN)
Assuming p-bit octal [1] *numeral is employed to construct a PIN, only one out of those* 8^p *possible combinations is the secret PIN. No one except for the owner of the PIN knows it. Then question is, "How many average trials-and-errors will be needed for a non-owner to know the PIN under a specific strategy?"*

This might be reminiscent of the famous problem called *a needle in a haystack* which was originally proposed by Hinton & Nowlan in 1987 [1]. The needle in the proposal was exactly the one configuration of 20-bit binary string, that is, the search space is made up of 2^{20} points and only one point is the needle to be searched for. No information such as how close is a currently searching point to the needle, or how likely is a searching point to be the needle. See Figure 1.

We assume that TCP connections to a computer network are represented with n-dimensional vectors and those represented by intrusions are such as *needles* among huge amount of normal transactions which might look like a *haystack* or *pastoral*.

[1] You will see the reason why *"octal"* not *"decimal"* later in the subseciton concerning *"intron"* in the section EXPERIMENTS.

II. Network Intrusion Detection

Those highly qualified hackers who provide security services to companies during the daytime and then go home at night to conduct totally illegal hacking are the ones who are the most dangerous. – by Enis Senerdem from Turkish Daily News on 29 March 2006.

When we are to design a network intrusion detection system, which is one of the hottest topics these days, by means of so-called a *soft computing* such as *artificial immune system*, *fuzzy logic*, *evolutionary computations*, *neural networks*, whatever it might be, we need a set of sample data to train the system and to test the system afterwards.

A. When a Family of Iris Flower is Normal Then are Others Abnormal? — Where is an Outlier?

The Spearman's iris flower database[2] is a frequently used dataset in pattern recognition/classification, data mining, etc. As such, there have been fair amount of studies in which this iris flower database is employed as a dataset to train and to test the intrusion detection system.

A total of 150 samples consists of three species: *setosa*, *versicolor* and *virginica*, each of which includes 50 samples. Each sample is a four-dimensional vector representing four attributes of the iris flower, that is, *sepal length*, *sepal width*, *petal length*, and *petal width*.

Let us take an example where this iris flower dataset was employed. Castellano et al. [2] assumed one family to be normal whilst the other two to be abnormal. The whole dataset was divided into 10 parts each of which has 15 samples uniformly drawn from the three classes. The system

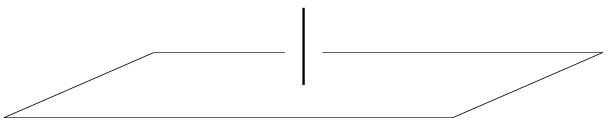

Fig. 1. A fictitious sketch of fitness landscape of *a needle in a haystack*. The haystack here is drawn as a two-dimensional flat plane of fitness zero.

[2] University of California Urvine Machine Learning Repository. ics.uci.edu: pub/machine-learning-databases.

is trained by the remaining 135 samples. The originally picked up 15 samples are used to test the results. After this 10-fold cross validation, the authors concluded that the abnormal detection rate is 96% while the false alarm rate is 0.6%. How nice, isn't it? In reality, however, it is not so simple. It might not be difficult at all for a hacker to find an unlearned region which could work to invade the system.

We now look at the Figure 2 to see how the three species are distributed in the whole search space. This is depicted by the Sammon Mapping.

Sammon Mapping maps a set of points in a high-dimensional space to the 2-dimensional space with the distance relation being preserved as much as possible, or equivalently, the distances in the n-dimensional space are approximated by distances in the 2-dimensional space with a minimal error.

Just a brief look at the figure reveals us that there remains an enormously wide region of unlearned for outliers.

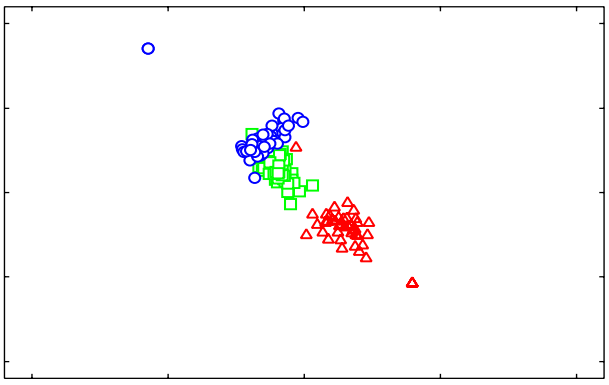

Fig. 2. A 2-D visualization of iris flower data by Sammon Mapping. Three different families of iris flower each contains 50 samples are represented in the figure with circles, triangles and squares.

B. Intrusion Might Look Like a Needle in a Hay!

The other type of dataset, naturally more often employed in the context of network intrusion, is the KDD cup 1999 dataset which was prepared by MIT Lincoln Laboratory as a dataset for the 1998 DARPA intrusion detection evaluation [3]. This dataset has been, and still is going to be, a common benchmark for evaluation of intrusion detection techniques.

KDD dataset, beside data for *Normal*, covers four major categories of *attacks*: (i) *Probing* attacks which attack by proving a vulnerability of the network; (ii) *Denial-of-Service (DoS)* attacks which try an invasion by denying legitimate requests to a system; (iii) *User-to-Root (U2R)* attacks which tries an unauthorized access to local superuser or root; and (iv) *Remote-to-Local (R2L)* attacks which is an unauthorized local access from a remote machine. These four categories of attacks include a total of 32 different attack types.

The dataset consists of two sub-datasets. The one is provided as *training* data and contains 4,898,430 records each of which is labeled as either normal, or attack indicating one specific attack out of the 32 types.[3] The second is unlabeled and contains 311,029 records, which is provided as *testing* data.

What a huge dataset! In fact, the Sammon Mapping we had tried in the iris dataset above wouldn't work any more. Therefore, many have tried various approaches to reduce the dimension. Let's start our small literature survey with this topic of *dimension reduction.*

Kuchimanchi et al. [4] used the *principal component analysis* (PCA), and calculated the first most important 19 attributes.[4] Then they evaluated the result of this dimension reduction by providing both the original 41-dimensional data and those 19-dimensional data reduced by PCA to a *decision-tree-classifier*, comparing *detection accuracies* and *false positive*[5] rates. They showed *detection accuracy* and *false positive* rate were 99.92% and 0.26%, respectively, on the 19-dimensional PCA data, while 99.94% and 0.23%, respectively, on the original 41-dimensional data.[6] What a successful result! However, is this still very huge, is it not?.

Let's see one more example. Joshi et al. [5] wrote, *"Exploiting only 5 out of 41 attributes*[7] *the best results was 79% accuracy in correctly detecting attacks, and 21% is accounted for false positive rate plus false negative*[8] *rate."* [9]

Though it might not be so successful as the above result by Kuchimanchi et al., if we consider *5 out of 41 attributes*, it is amazing. Wow!

Anyway, it is good to know we can reduce the dimension of the original data set of KDD cup 1999 dataset into at least about half with the result being intact.

Then, our next interest will be, "Are all of the attack types in the KDD cup 1999 dataset are equally willing to wait to

[3] The labeled training dataset includes 972,780 Normals, 41,102 Probes, 3,883,370 DoSs, 52 U2Rs, and 1,126 R2Ls.

[4] They are *src_bytes, dst_bytes, duration, is_guest_login, is_host_login, srv_diff_host_rate, diff_srv_rate, service, flag, protocol_type, num_root, hot, num_compromised, dst_host_same_srv_rate, dst_host_count, rerror_rate, srv_count*, and *dst_host_srv_diff_host_rate.*

[5] I.e., recognizing attack as normal.

[6] This was not the main purpose of the paper. The authors rather exploited the other methods of dimension reduction such as *neural-network-PCA* or *nonlinear-component-analysis*, expecting more efficiency and higher accuracy. The evaluation was carried out not only by *decision-tree classifier* but also by *non-linear classifier.*

[7] I.e., src_bytes, dst_bytes, duration, is_host_login, and is_guest_login.

[8] I.e., recognizing normal as attack.

[9] Most of the phrases cited in this article hereafter like *"···"* were paraphrased, more or less, by the author of this article. As such, if there are some incorrect expressions, it is the author of this article who is responsible for, not the original authors.

be detected?" Some of the reports were from this point of view. Let us name a few.

Pan et al. [6] exploited *three-layer (70-14-6) feed-forward neural network* with a sigmoid transfer function trained with back-propagation using scaled conjugate gradient decent, to detect five typical types of attacks (neptune, portsweep, satan, buffer_overflow, and guess_passwd) as well as normal samples.

Let's see what they observed. Authors wrote, *"The test result indicates that 99.6% of the normal examples were recognized correctly, and for three attacks of neptune, satan, portsweep, we obtained the average detect rate of 96.6% and the false positive rate of 0.049%. However, for all the five kinds of attacks, we only obtained the average detect rate of 64.9% and the false positive rate of 26.7%. This is because all buffer_overflow and guess_passwd attacks failed to be classified by this back-propagation neural network. Then we tried an expert system, and found that the R2L and U2R can be more accurately detected by this rule-based detector than neural network."* And then concluded, *"The model based on both neural network and expert system finally achieved the average detection rate of 93.28% and false positive rate of 0.2% for all of these five types of attacks."*

We, however, would be rather more interested in why this neural network failed to classify *buffer_overflow* and *guess_passwd* attacks, rather than the performance improvement by using rule-based detector.

Pan et al. reported yet another result in their different article [7]. With the same architecture of neural network and with the same target of five attacks as above, but using C4.5 instead of expert system, they reported that correctly predicted (normal, neptune, satan, portsweep, buffer_overflow, guess_passwd) was (73.3%, 99.2%, 94.6%, 94.2%, 0.0%, 0.0%). And concluded, "*The back-propagation network can't detect the buffer_overflow and guess_passwd attacks."* This sounds like a realistic assertion, and the one we want.

Thus far, such more careful conclusions appear in the recent literatures. For example, Stibor et al. [8] wrote, *"The real-valued negative selection with variable-sized detectors has poor classification performance on the high-dimensional KDD dataset."*

When this artificial immune system based detector was proposed by Ji et al. [9], the result of applying it to the *iris dataset* was not that bad. That is, the correct detection rate of (*setosa*, *versicolor*, *virginica*) was (99.98%, 85.95%, 81.87%), while false alarm rates were all zero!

As another example of such *implicit* report of failure, Dam et al. [10] claimed, *"The evolutionary classifier system, devised to make its performance improved than the traditional one, resulted in the detection rate of (95.7%, 49.1%, 93.0%, 8.5%, 3.9%) for (normal, DoS, Probe, U2R, R2L)."*

Again, we are rather more interested in why detection rate is so low for U2R and R2L than whether result is satisfactory or not.

Finally, it would be interesting to take a look what Sabhnani et al. [11] reported. See Table 1 to have a bird's eye view of those results above.

Table 1. Detection rate for 4 attack types each with 9 different machine learning technique. From Sabhnani et al. [11].

	Probe	DoS	U2R	R2L
Multilayer Perceptron	88.7	97.2	13.2	5.6
Gaussian Classifier	90.2	82.4	22.8	9.6
K-mean Clustering	87.6	97.3	29.8	6.4
Nearest Cluster Algorithm	88.8	97.1	2.2	3.4
Radial Basis Function	93.2	73.0	6.1	5.9
Leader Algorithm	83.8	97.2	6.6	1.0
Hypersphere Algorithm	84.8	97.2	8.3	1.0
Fuzzy Art Map	77.2	97.0	6.1	3.7
C4.5 Decision Tree	80.8	97.0	1.8	4.6

Also note that KDD cup 1999 winner's detection rate for (Probe, DoS, U2R, and R2L) was (83.3%, 97.1%, 13.2%, 8.4%).

Our Conjecture

Here, we, conjecture that those sometimes observed poor results are because some of the attack data are like needles in a haystack of huge amount of normal data. If we were able to fully visualize such large size of normal samples together with a few data picked up from abnormal samples, the latter might look like a needle in a hay stack of the former, like in Figure 3. Though we are not yet ready, we plan to show a visualization of this assumption of us elsewhere, to study this conjecture further in detail.

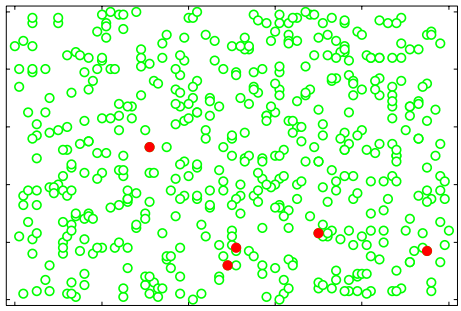

Fig. 3. We conjecture that some attack data (filled circles) are like needles in a hay of normal data (empty circles). Plots in this figure are all fictitious.

To summerize this section, we ask the readers,

Problem 2 (A Challenge in KDD-cup-99 dataset) *Design an intrusion detection system which has 41 inputs corresponding to attributes from KDD-cup-1999 dataset, and 5 YES/NO outputs indicating that the input is either normal, Probe, DoS, U2R, or R2L. The question is, "Such design is possible or not?"*

Also see Figure 4 to get an image of real implementation by a neural network, as an example.

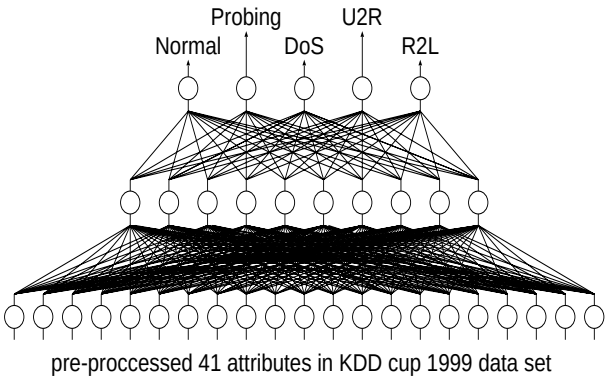

Fig. 4. A simple architecture of neural network we desire to design to classify KDD cup 99 dataset.

III. Experiments

Flick remembered the legend of the Jackdaw of Rheims, the bird that stole the bishop's ring. The monks couldn't figure out who had taken it, so the bishop cursed the unknown thief. Next thing they knew, the jackdaw appeared all bedraggled, and they realized he was suffering from the effects of curse, and must be the culprit. Sure enough they found the ring in his nest. – "Jackdaws" by Ken Follett.

Assuming our conjecture that real attack sample is like a needle in a haystack of normal samples, we now look at how easy or difficult to find it? Let's start with a random search. Note that some proposed algorithms which were reported as success actually were not good as asserted, and sometimes found to be worse than a random search.

A. Random Fall of Parachutists

Algorithm 1 (Random Fall) *(1) Create a p-bit octal PIN at random. (2) Create randomly one* $3p$*-bit of binary string. (3) Translate the string into p-bit octal code. (4) Check if the translated code matches the PIN. (5) If matches end the run. Otherwise go back to 2.*

Let us allow to use a metaphor here. We now assume only one needle in a pastoral, and parachutists fall from the airplane in the sky to the pastoral one by one, then how would it be likely for a parachutist to fall just on the needle. This might be taken as a random search, and will be our criterion of comparison hereafter.

Note that one parachutist is represented by our genotypes of a $3p$-digit binary strings. Let's take an example of $p = 4$. A genotype

$$((100)(111)(000)(010))$$

maps into its phenotype (4 7 0 2).

At the start, one p-bit octal PIN is created which we assume no one knows *a priori*. With p being increasing from 2, we count the number of randomly created genotypes until its phenotype strictly matches to the hidden PIN. The average number, during 1024 runs, of parachutists needed until we found the parachutist who fell on the needle just by chance, for $p = 2, 3, 4, 5, 6, 7$ were, respectively,

$$(66, 512, 3951, 32154, 254673, 2058527). \quad (1)$$

See Figure 5.

B. What if Parachutists are Allowed to Walk after Fall?

Algorithm 2 (Random Fall) *(1) Create a p-bit octal PIN at random. (2) Create randomly one* $3p$*-bit of binary string. (3) Translate the string into p-bit octal code. (4) Check if the translated code matches the PIN. (5) If matches end the run. Otherwise give a mutation by flipping a bit chosen at random*[10] *with a probability of* $1/3p$ *until the translation matches the PIN, or number of steps exceeds 1000. (6) If still hasn't been matched, then repeat from 2.*

What will happen if the fallen parachutist is allowed to explore, say, 1000 random steps, around the falling spot? This might remind readers of the seminal experiment once made by Hinton & Nowlan [1] who referred it to *"lifetime learning — Baldwin Effect"* though our parachutist in this paper ends her life without creating a next generation. The results, again over the average of 1024 runs, for $p = 4, 5, 6, 7, 8$ were, respectively,

$$(5, 36, 308, 2436, 23087). \quad (2)$$

The results are depicted also in Figure 5 with the result of our random parachutists in the previous subsection. In both cases, we can see that complexity to find the needle is an exponential order. But look! How impressive an *exploration-after-fall* improves the performance!

As you have probably noticed already, however, it's not fair just to compare the number of parachutists. The total number of points searched by those walking parachutists is plotted as a function of p in Figure 6. We can see that the result was rather worse than our random parachutists, despite of its superficial good looking of the result.

[10] We will call this a *"point-wise mutation"* hereafter.

C. Neutral Mutation

Algorithm 3 (Walk by Neutral Mutations) *(1) Create a PIN at random. (2) Create one genotype at random. (3) Try point-wise mutation on the genotype such that the result maps into the same phenotype as the one before the mutation. (4) Assess all possible single-mutation-neighbors of the new genotype to determine whether any new phenotypes were discovered. (5) Step 3 to 4 are repeated untill the phenotype matches the PIN, or untill a pre-fixed number of steps is reached.*

This is a paraphrase of the algorithm proposed by Shipman et al. [12] who called the step 3 a *neutral mutation* (Note that the mutation in step 4 is a standard one). Its efficiency was studied in their paper by applying it to a random Boolean network and telecommunication networks. But why not more simple example is to be explored, if it is to work universally?

To apply this in our problem of searching for the needle, that is, octal p-bit PIN, we design our genotype as $15p$-bit binary string such that the *number-of-1 (mod 8)* in each of those 15-digit blocks in the string maps into one bit of the corresponding octal code.[11] For example

((100011000000100)(111111111111111)(111110001101010))

maps into (4 7 1).

The average number, during 1024 runs, of parachutists needed until we found the one who firstly reached the needle for $p = 2, 3, 4, 5, 6$ were, respectively,

$$(71728, 583593, 4930624, 36592634, 314817878). \quad (3)$$

Much worse than our random search.

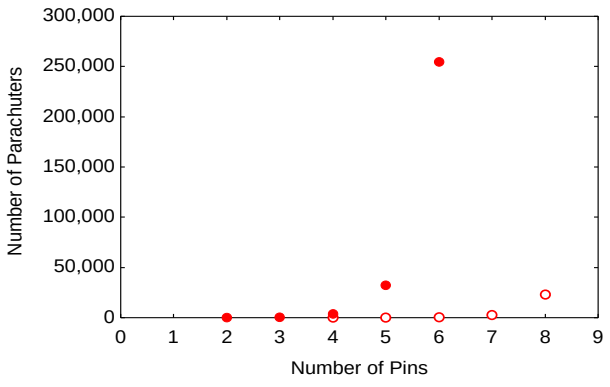

Fig. 5. (a) Number of random creations of candidate until the PIN is matched (filled circles), and (b) The number when created candidate is allowed random walks of 1000 steps (empty circles). Both are the average of 1024 runs.

[11] A simple consideration might give us the idea that 7-bit binary for each octal is enough. However, we implemented in this way so that each octal from 0 to 7 are created uniformly at random.

D. Does Neutral Mutation on Intron Enhance Efficiency of Search?

Algorithm 4 (Neutral Network) *(1) Create a PIN at random. (2) Create randomly an initial individual which is considered to be the winner to the next generation. (3) Carry out point-wise mutation on the winning parent to generate 4 offspring. (4) Construct a new generation with the winner and its offspring. (5) Select a winner from the current population using the following rules. (i) If any offspring has a better fitness than the parent, the one with highest fitness becomes the winner. (ii) If fitness of all offspring have the same fitness as the parent, one offspring is randomly selected, and if the parent-offspring pair has a Hamming distance within the permitted range, the offspring becomes the winner, otherwise the parent remains as the winner. (5) Back to step 2 unless the maximum number of generations reaches, or a solution is found.*

The description of the algorithm above is a paraphrase from Yu & Miller [13]. We had an interesting discussion between Yu & Miller's paper *"Finding needles in haystack is not hard with neutrality"* (2002) vs. Collin's *"Finding needles in haystack is harder with neutrality"* (2005).

What Yu & Miller [13] attacked as a type of *a needle in a haystack* problem was to make a genetic algorithm construct a even-n-parity logic circuit by employing only *XORs* and *EQs*, not *ANDs* and *ORs* and so on, which shows a peculiar fitness landscape. The even-n-parity logic has n-bit binary inputs and if and only if the number of "1" is even, it returns 1 and otherwise returns 0. Hence, we can evaluate the fitness value of any one candidate of the solution, by giving all the possible configurations of 0 and 1 and counting how many correct outputs. Thus, from a combinatorial point of view we have 2^n cases of fitness values. In reality, however, we have, only three different values, that is, 2^n, $2^{(n-1)}$ and 0. In other words, the output is all correct, half correct, or not correct at all. For example, candidates of even-3-parity constructed only by *XORs* and *EQs* returns either 8, 4 or 0 correct outputs for

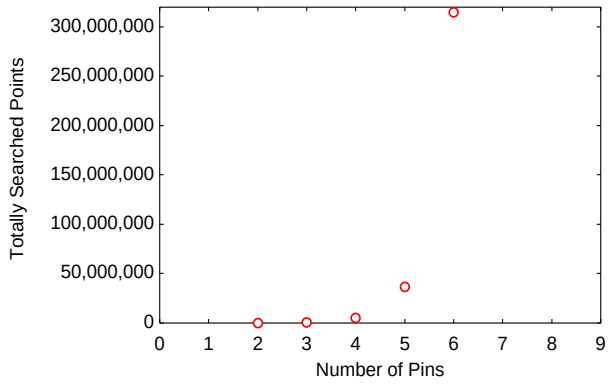

Fig. 6. Average number of points explored by all the randomly created candidates who are allowed further random walks of 1000 steps until the needle is found. Average are taken among 1024 runs.

the eight possible inputs (000), (001), (010), ..., (111).

Yu & Miller wrote, *"In the case of random creation of 4,000,000 candidates of even-12-parity, the solution (fitness 4096) was never emerged, while even-10-parity 100,000 random creations of candidate yielded 540 solutions (fitness 1024). On the other hand, when neutral mutation was applied to the candidates of even-12-parity, the 48 out of 100 runs reached solution(s) with each run being only within 10,000 iterations."*

Collins argued back concluding, *"Reported success is due to a bias of the selection"* [14]. In the other Collin's work [15], it was analytically shown that the number of possible candidates of even-12-parity is 1.315×10^{139} in which number of real solutions is 2.568×10^{132}, claiming *"Yu & Miller's result is, therefore, worse than a possible random search."*

Again what we want to emphasize here is, if the assertion by Yu & Miller is universally true, it would work in yet more principally simple examples. Before going to proceed, let's see what is *intron* that Yu & Miller assumed to play an important role in their evolution. For example, take a look at a genotype representing an even-3-parity,

$$((EQ, A, B)(EQ, C, D)(XOR, 1, E)(EQ, F, G)(EQ, 3, H))$$

where each gene which corresponds to one unit constructs triples, with the 1st being which logic to be used (EQ or XOR); and with the 2nd and 3rd being connections to either one of the inputs or the outputs of a previous unit. Note that the 2nd and 4th genes in the above example do not contribute to construct the phenotype since those two genes will not be connected to any other unit, and hence are called *intron* as a biological metaphor. Any mutation on an intron has no effect on phenotype, and as such, they are called *neutral*. The above genotype can be interpreted as the phenotype shown in Figure 7.

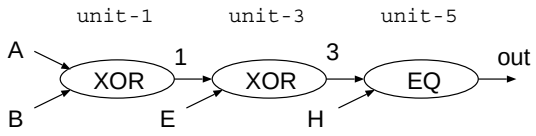

Fig. 7. An example of phenotype of even-3-parity constructed only by EQs and XORs.

Now we try to apply this to our finding PIN problem. This time we use 4-digit binary, instead of 3-digit as before, to represent one octal numeral in the candidate of PIN. Then translation is into decimal, instead of octal, and when the translated decimal is larger than seven we consider it an *intron*. For example

$$((0001)(1100)(0101)(0010)(0111))$$

is translated into $(1, 5, 2, 7)$ since the second gene is translated into 12 and supposed to be an *intron*.

The average number, during 1024 runs, of parachutists needed to firstly find the needle for $p = 2, 3, 4, 5$ were, respectively,

$$(65, 488, 3751, 33710). \quad (4)$$

Alas, if we compare it with (1) we will see that the result is almost the same as our random parachutists.

Possible Conclusion
We have no such algorithm that can more efficiently look for a needle in a haystack than a random search. No way to find needles in a pastoral.

IV. Discussion

As Laskov et al. [16] claimed in their paper, *"Labels can be extremely difficult or impossible to obtain. Analysis of network traffic or audit logs is very time-consuming and usually only a small portion of the available data can be labeled. Furthermore, in certain cases, for example at a packet level, it may be impossible to unambiguously assign a label to a data instance."* Authors further wrote, *"In a real application, one can never be sure that a set of available labeled examples covers all possible attacks. If a new attack appears, examples of it may not have been seen in training data."* Then our next question is,

Problem 3 (Attacks by Mutants) *Pick up at random a set of n normal samples from KDD cup 1999 dataset. All of those n samples are given a point-wise mutation and taken as attack data. Train your intrusion detection system using half of the normal samples and half of the attack samples (the number of both is n/2), then test the system using the remaining samples. Can the system detect those mutants as intrusion?*

A. Can a Sommelier be Trained without Bootlegs?

Though we have not remarked so far, there remains further difficult issue, that is, "How the system can learn only from normal data to detect abnormal?" We usually have enormous amount of normal data but we have no information about coming attacks untill it's too late.

Gomez et al. [17] claimed, *"A new technique for generating a set of fuzzy rules that can characterize the non-self (abnormal) space using only self (normal) samples."* Their experiment employed 10% dataset, also given as a part of KDD cup 1999 dataset, which reduced the number of records into 10% of the original ones. Further, they removed categorical attributes and normalized these remaining 33 numerical attributes between 0 and 1 using the maximum and minimum values found. Then 80% of the normal samples were picked up at random for training while the remaining 20% along with the same number of abnormal samples were used for testing. Gomez et al. designed

the detector with what they called an *"immuno-fuzzy approach"* and system they call an *"evolving fuzzy rules detectors"* claiming, *"It detects attacks with the detection rate 98.30% and false alarm rate 2.0%."* Really satisfactory, if it's really true.

The report didn't mention about the categories of attacks, which implies the reported success is an average over all attack types. It seems to be too good if we consider the results of other not-so-happy reports mentioned above.

More important thing to notice here is the system learned from *"only with normal data"* to establish this success. It would be terrific if it was really true, but we are fishy more or less.

This issue is something like we require a wine-taster to recognize bootleg champagne by only providing him/her plenty of real champagne to learn.[12] Though this *training-only-with-normal* is our ultimate goal, but not so simple to be realized. To study how this is difficult, why not try the following?

Problem 4 (Dummy Attacks) *(1) Prepare two sub-datasets from KDD cup 1999 dataset. One is picked up from normal samples and call it* D_{normal}*. The other is from attack samples and call it* D_{attack}*. (2) Furthermore, randomly create an attack dataset – dummy attacks, and call it* D_{dummy}*. (3) Train your intrusion detection system only with* D_{normal}*. (4) Then, try two tests, one with only* D_{attack}*, and the other with only* D_{dummy}*, avoiding any a priori prediction.*

B. Don't We Expect the Result a priori?

"Artificial immune system detects an attack by computer viruses!" How fantastic it sounds. Whilst we wish it would work, we are afraid it might be just a *fantasy.* So, we need a placebo experiment.

Of the 311,029 records in the test set of KDD cup 1999, the rate of (Normal, Prove, DoS, U2R, R2L) is (19.5%, 1.3%, 73.9%, 5.2%, 0.1%), respectively. This suggests that even the *always-return-U2R strategy*[13] for instance, would result in the accurate detection rate of (Normal, Probe, DoS, U2R, R2L) = (0.0%, 0.0%, 0.0%, 5.2%, 0.0%). Or, the *always-return-a-random-output strategy*[14] would have quite a high score to detect DoS attacks.

The two strategies above might be more intelligent than some of the *artificial intelligent* techniques so far proposed, rather than *ignorant.*

We have to be careful, because we sometimes tend to *unconsciously* pick up only a set of data that is suitable to draw our conclusion which was *a priori* expected, if not *intentionally* at all.

In the way that just a *powder-from-sugar* sometimes has a same effect as, or more efficient than, a medicine under developing enough to cure a disease for a group of innocent volunteers. Let's conclude with the following final question.

Problem 5 (Placebo Experiment) *(1) Create a simple device which randomly returns either of Normal, Prove, DoS, U2R, or R2L for any input. (2) Prepare a test dataset including enough amount of records uniformly from Normal, Prove, DoS, U2R, and R2L. (3) Compare the performances of the detector you designed with the random-reply-machine created in step 1, feeding the same dataset prepared in step 2.*

V. Concluding Remarks

As we have described so far, KDD cup 1999 intrusion detection dataset has 4,898,430 records in the labeled dataset for *training* purposes of which 75.6111% are normal. On the other hand, we have 311,029 records in the unlabeled dataset for *testing* purposes of which only 0.0733% are U2R, for example. Under this situation, a likely interpretation would be the U2R attack patterns are like needles in hay of normal patterns when it undergoes a test, if we are not very lucky. Considering we have not had so satisfactory results to detect U2R attacks, we do not seem to be so lucky.

In addition, if we take it account that a hacker is a person who is extremely good at finding a pattern which is very close to the normal traffic, the point that might be located by a hacker is not a randomly located point.

It is said that we have two kind of intrusion detection. One is called *misuse* detection which recognizes known attack patterns. The other is called *anomaly* detection which detects no-normal unknown patterns. We are not interested in the former. All we want is to detect unknown outliers. And an *outlier* usually lies not far from normal but very close to it. We could not be so optimistic.

As for using an *artificial immune system*, for example, since that real sensational proposition in 1994 by Forrest, Perelsen et al. [18] that claimed, *"Negative selection of a metaphor of our real biological immune system can detect anomaly as non-self in computers,"* we have had tremendously lots of intelligent challenges for more than two decades, but all in vain in a real sense. Still this topic is not fruitful at all from a practical point of view, as far

[12] Or, in an opposite way. I usually enjoy Georgian sparkling wine like once a week, but still a real champagne would be able to pretend to be a Georgian one to me."

[13] which returns U2R whatever the input is.

[14] which returns either Normal, Probe, DoS, U2R, or R2L at random regardless of the input.

as we know.

Probably the most intelligent way of detecting a network intrusion is to *curse it and wait for the effect of the curse.*

Needless to say, however, this article is not to negate the possibility, but we hope this will be a serious challenge to intrusion detection community to emerge real innovative ideas.

References

[1] G. E. Hinton, and S. J. Nowlan (1987) *"How Learning can Guide Evolution."* Complex Systems, 1, pp. 495–502.

[2] G. Castellano, and A. M. Fanelli (2000) *"Fuzzy Inference and Rule Extraction using a Neural Network."* Neural Network World Journal, Vol. 3, pp. 361– 371.

[3] S. J. Stolfo, F. Wei, W. Lee, A. Prodromidis, and P. K. Chan (1999) "KDD Cup knowledge discovery and data mining competition." http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[4] G. K. Kuchimanchi, V. V. Phoha, K. S. Balagani, and S. R. Gaddam (2004) *"Dimension Reduction Using Feature Extraction Methods for Real-time Misuse Detection Systems."* Proceedings of Workshop on Information Assurance and Security, pp. 1555–1563.

[5] S. S. Joshi, and V. V. Phoha (2005) *"Investigating Hidden Markov Models Capabilities in Anomaly Detection."* Proceedings of the 43rd ACM Southeast Conference, Vol. 1, pp. 99–103.

[6] Z. Pan, H. Lian, G. Hu, and G. Ni (2005) *"An Integrated Model of Intrusion Detection Based on Neural Network and Expert System."* Proceedings of IEEE International Conference on Tools with Artificial Intelligence, pp. 671–672.

[7] Z. Pan, S. Chen, G. Hu, and D. Zhangn (2003) *"Hybrid Neural Network and C4.5 for Misuse Detection."* Proceedings of International Conference on Machine Learning and Cybernetics, pp. 2463–2467.

[8] T. Stibor, J. Timmis, and C. Eckert (2005) *"A comparative Study of Real-valued Negative Selection to Statistical Anomaly Detection Techniques."* Proceedings of International Conference on Artificial Immune Systems, Lecture Notes in Computer Science, Vol. 3627, Springer, pp. 262–275.

[9] Z. Ji, and D. Dasgupta (2004) *"Real-valued Negative Selection Algorithm with Variable-sized Detectors."* Proceedings of Genetic and Evolutionary Computation Conference, Lecture Notes in Computer Science Vol. 3102, Springer, pp. 287–298.

[10] H. H. Dam, K. Shafi, and H. A. Abbass (2005) *"Can Evolutionary Computation Handle Large Dataset?"* Technical Report: The Artificial Life and Adaptive Robotics Laboratory, TR-ALAR-200507011.

[11] M. Sabhnani, and G. Serpen (2003) *"Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context."* Proceedings of the International Conference on Machine Learning: Models, Technologies and Applications, pp. 209–215.

[12] R. Shipman, M. Shackleton, and I. Harvey (2000) *"The Use of Neutral Genotype-phenotype Mappings for Improved Evolutionary Search."* BT Technology Journal, Vol. 18, No. 4, pp. 103–111.

[13] Tina Yu and J. Miller (2002) *"Finding Needles in Haystacks is Not Hard with Neutrality."* Proceedings of EuroGP 2002, Lecture Notes in Computer Science Vol. 2278, Springer, pp. 13–25.

[14] M. Collins (2005) *"Finding Needles in Haystacks is Harder with Neutrality."* Proceedings of Genetic and Evolutionary Computation Conference, pp. 1613–1618.

[15] M. Collins (2004) *"Counting Solutions in Reduced Boolean Parity."* CD-ROM of Workshop Proceedings in Genetic and Evolutionary Computation Conference.

[16] P. Laskov et al. (2005) *"Learning Intrusion Detection: Supervised or Unsupervised?"* Proceedings of International Conference on Image Analysis and Processing, Lecture Notes in Computer Science, Vol. 3617, Springer, pp. 50–57.

[17] J. Gomez, F. Gonzalez, and D. Dasgupta (2003) *"An Immuno-Fuzzy Approach to Anomaly Detection."* Proceedings of IEEE International Conference on Fuzzy Systems, Vol. 2, pp. 1219-1224.

[18] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri (1994) *"Self Nonself Discrimination in a Computer."* Proceedings of IEEE Symposium on Research in Security and Privacy, pp. 202– 212.