A Linear Genetic Programming Approach to Intrusion Detection

Dong Song, Malcolm I. Heywood, and A. Nur Zincir-Heywood

Dalhousie University, Faculty of Computer Science
6040 University Avenue, Halifax, NS, B3H 1W5, Canada
{dsong,mheywood,zincir}@cs.dal.ca

Abstract. Page-based Linear Genetic Programming (GP) is proposed and implemented with two-layer Subset Selection to address a two-class intrusion detection classification problem as defined by the KDD-99 benchmark dataset. By careful adjustment of the relationship between subset layers, over fitting by individuals to specific subsets is avoided. Moreover, efficient training on a dataset of 500,000 patterns is demonstrated. Unlike the current approaches to this benchmark, the learning algorithm is also responsible for deriving useful temporal features. Following evolution, decoding of a GP individual demonstrates that the solution is unique and comparative to hand coded solutions found by experts.

1 Introduction

The Internet, as well as representing a revolution in the ability to exchange and communicate information, has also provided greater opportunity for disruption and sabotage of data previously considered secure. The study of intrusion detection systems (IDS) provides many challenges. In particular the environment is forever changing, both with respect to what constitutes normal behavior and abnormal behavior. Moreover, given the utilization levels of networked computing systems, it is also necessary for such systems to work with a very low false alarm rate [1]. In order to promote the comparison of advanced research in this area, the Lincoln Laboratory at MIT, under DARPA sponsorship, conducted the 1998 and 1999 evaluation of intrusion detection [1]. As such, it provides a basis for making comparisons of existing systems under a common set of circumstances and assumptions [2]. Based on binary TCP dump data provided by DARPA evaluation, millions of connection statistics are collected and generated to form the training and test data in the Classifier Learning Contest organized in conjunction with the 5th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining 1999 (KDD-99). The learning task is to build a detector (i.e. a classifier) capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" or normal connections. There were a total of 24 entries submitted for the contest [3,4]. The top three winning solutions are all variants of decision trees. The winning entry is composed from 50×10 C5 decision trees fused

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 2325–2336, 2003.

by cost-sensitive bagged boosting [5]. The second placed entry consisted of a decision forest containing 755 trees [6]. The third placed entry consisted of two layers of voting decision trees augmented with human security expertise [7].

In this work, the first interest is to explore a Genetic Programming approach to produce computer programs with a much less complex structure, compared with the above data mining approaches, yet yielding satisfactory performance on the KDD-99 test set. The motivation being to provide transparent solutions that execute in real time with modest computational resources. Second, the training system must scale well on a comparatively large training data set (the 10% KDD-99 training set consists of approximately half a million patterns) whilst providing a small enough computational footprint to complete training in a matter of hours on a personal computer. Third, genetic programs are translated into human readable format and are compared with human reasoning. This will facilitate a better understanding of what constitutes a good rule as well as gaining the confidence of the more technically motivated users. Finally, only the most basic feature set is to be utilized, as opposed to the 41 connection features used in the decision tree solutions. To this end, a Page-based Linear Genetic Program with Dynamic Subset Selection and Random Subset Selection is proposed. Forty trials are conducted and results are compared with KDD-99 winning entries. One individual was simplified by removing structural introns and analyzed. The solution was found to be comparative to the type of rules extracted by domain experts on the same data set.

In the following text, Section 2 summarizes the properties associated with the KDD-99 IDS data set. Section 3 details the Genetic Programming (GP) approach taken, with a particular emphasis on the methodology used to address the size of the data set. Section 4 describes parameter settings and evaluates experiment results. Finally, conclusions and future directions discussed in Section 5.

2 Intrusion Detection Problem

From the perspective of the Genetic Programming (GP) paradigm KDD-99 posted several challenges [1, 2]. The amount of data is much larger than normally the case in GP applications. The entire training dataset consists of about 5,000,000 connection records. However, KDD-99 provided a concise training dataset – which is used in this work – and appears to be utilized in the case of the entries to the data-mining competition [3-7]. Known as "10% training" this contains 494,021 records among which there are 97,278 normal connection records (i.e. 19.69 %). This will be addressed by using a hierarchical competition between patterns, based on the Dynamic Subset Selection technique [8].

Each connection record is described in terms of 41 features and a label declaring the connection as either normal, or as a specific attack type. Of the 41 features, only the first eight (of nine) "Basic features of an Individual TCP connection"; hereafter referred to as 'basic features' are employed by this work. The additional 32 *derived* features, fall into three categories,

Content Features: Domain knowledge is used to assess the payload of the original TCP packets. This includes features such as the number of failed login attempts;

Time-Based Traffic Features: These features are designed to capture properties that mature over a 2 second temporal window. One example of such a feature would be the number of connections to the same host over the 2 second interval;

Host-Based Traffic Features: Utilize a historical window estimated over the number of connections – in this case 100 – instead of time. Host based features are therefore designed to assess attacks, which span intervals longer than 2 seconds.

In this work, none of these additional features are employed, as they appear to almost act as flags for specific attack behaviors. Our interest is on assessing how far the GP paradigm would go on 'basic features' alone.

Thirdly, the training data encompasses 24 different attack types, grouped into one of four categories: User to Root; Remote to Local; Denial of Service; and Probe. Naturally, the distribution of these attacks varies significantly, in line with their function – 'Denial of Service,' for example, results in many more connections than 'Probe'. Table 1 summarizes the distribution of attack types across the training data. Test data, on the other hand, follows a different distribution than in the training data, where this has previously been shown to be a significant factor in assessing generalization [3]. Finally, the test data added an additional 14 attack types not included in the training data.

Table 1. Distribution of Attacks

Data Type	Training	Test
Normal	19.69%	19.48%
Probe	0.83%	1.34%
DOS	79.24%	73.90%
U2R	0.01%	0.07%
R2L	0.23%	5.2%

Given that this work does not make use of the additional 32 derived features, it is necessary for the detector to derive any temporal properties associated with the current pattern, $x(t)$. To this end, as well as providing the detector with the labeled feature vector for the current pattern, $[x(t), d(t)]$, the detector is also allowed to address the previous 'n' features at some sampling interval (modulo(n)).

3 Methodology

In the case of this work a form of Linearly-structured GP (L-GP) is employed [9-12]. That is to say, rather than expressing individuals using the tree like structure popularized by the work of Koza [13], individuals are expressed as a linear list of instructions [9]. Execution of an individual therefore mimics the process of program execution normally associated with a simple register machine. That is, instructions are defined in terms of an opcode and operand (synonymous with function and terminal sets respec-

tively) that modify the contents of internal registers {R[0],…,R[k]}, memory and program counter [9]. Output of the program is taken from register R[0] on completion of program execution (or some appropriate halting criterion [11]). Moreover, in an attempt to make the action of the crossover operator less destructive, the Page-based formulation of L-GP is employed [12]. In this case, an individual is described in terms of a number of *pages*, where each page has the *same* number of *instructions*. Crossover is limited to the exchange of *single* pages between two parents, and appears to result in concise solutions across a range of benchmark regression and classification problems. Moreover, a mechanism for dynamically changing page size was introduced, thus avoiding problems associated with the *a priori* selection of a specific number of instructions per page at initialization. Mutation operators take two forms. In the first case the 'mutation' operator selects an instruction for modification with uniform probability and performs an Ex-OR with a second instruction, also created with uniform probability. If the ensuing instruction represents a legal instruction the new instruction is accepted, otherwise the process is repeated. The second mutation operator 'swap' is designed to provide sequence modification. To do so, two instructions are selected within the same individual with uniform probability and their positions exchanged.

As indicated in the introduction, the specific interest of this work lies in identifying a solution to the problem of efficiently training with a large dataset (around half a million patterns). To this end, we revisit the method Dynamic Subset Selection [8] and extend it to the case of a hierarchy of subset selections. There are at least two aspects to this problem: the cost of fitness evaluation – the inner loop, which dominates the computational overheads associated with applying GP in practice; and the overhead associated with datasets that do not fit within RAM alone. In this work, a hierarchy is employed in which the data set is first partitioned into blocks small enough for retention in RAM, whilst a competition is initiated between training patterns within a selected block. Such a scheme also naturally matches the design methodology for computer memory hierarchies [14]. The selection of blocks is performed using Random Subset Selection (RSS) – layer 1. Dynamic Subset Selection (DSS) enforces a competition between different patterns – layer 2.

3.1 Subset Selection

First layer. The KDD-99 10% training data set was divided into 100 blocks with 5,000 connection records per block. The size of such blocks is defined to ensure that, when selected, they fit within the available RAM. Blocks are randomly selected with uniform probability. Once selected, a history of training pressure on a block is used to set up the number of iterations performed at the next layer in DSS. This is performed in proportion to the performance of the best-case individual. Thus, iterations of DSS, I, in block, b, at the current instance, i, is

$$I_b(i) \quad = \quad I_{(\max)} \quad \times \quad E_b(i - 1) \tag{5}$$

where $I_{(\max)}$ is the maximum number of subsets selected on a block; and $E_b(i-1)$ is the number of misclassifications of the best individual on the previous instance, i, of block, b. Hence, $E_b(i) = 1 - [\text{hits}_b(i) / \#\text{connections}(b)]$, where $\text{hits}_b(i)$ is the hit count

over block '*b*' for the best case individual identified over the last DSS tournament at iteration '*i*' of block '*b*'; and #connections(*b*) is the total number of connections in block '*b*'.

Second Layer. A simplified DSS is deployed in this layer. That is, fixed probabilities are used to control the weighting of selection between age and difficulty. For each record in the DSS subset, there is a 30% (70%) probability of selecting on the basis of age (difficulty). Thus, a greater emphasis is always given to examples that resist classification. DSS utilizes a subset size of 50, with the objective of reducing the computational complexity associated with a particular fitness evaluation. Moreover, in order to further reduce computation, the performance of parent individuals on a specific subset is retained. After 6 tournaments the DSS subset will be reselected.

DSS Selection [8]. In the RSS block, every pattern is associated with an age value, which is the number of DSS selections since last selection, and a difficulty value. The difficulty value is the number of individuals that were unable to recognize a connection correctly the last time that the connection appeared in the DSS subset. Connections appear in a specific DSS stochastically, where there is 30% (70%) probability to select by age (difficulty). Roulette wheel selection is then conducted on the whole RSS block, with respect to age (difficulty). After the DSS subset is filled, age and difficulty of selected connections are reset. For the rest, age is increased by 1 and difficulty remains unchanged.

3.2 Parameterization of the Subsets

The low number of patterns actually seen by a GP individual during fitness evaluation, relative to the number of patterns in the training data set, may naturally lead to 'over fitting' on specific subsets. Our general objective was therefore to ensure that the performance across subsets evolved as uniformly as possible. The principle interest is therefore to identify the stop criterion necessary to avoid individuals that are sensitive to the composition of a specific Second Level subset.

To this end, a single experiment is conducted in which 2,000 block selections are made with uniform probability. In the case of each block selection, there are 400 DSS selections. Before selection of the next block takes place, the best performing individual (with respect to sub-set classification error) is evaluated over *all* patterns within the block, let this be the block error at selection *i*, or $E_b(i)$. A sliding window is then constructed consisting of 100 block selection errors, and a linear least-squares regression performed. The gradient of each linear regression is then plotted, Figure 1 (1900 points). A negative trend implies that the block errors are decreasing whereas a positive trend implies that the block errors are increasing (the continuous line indicates the trend). It is now apparent, that after the first 750 block selections, the trend in block error has stopped decreasing. After 750 selections, oscillation in the block gradients appears, where this becomes very erratic in the last 500 block selections.

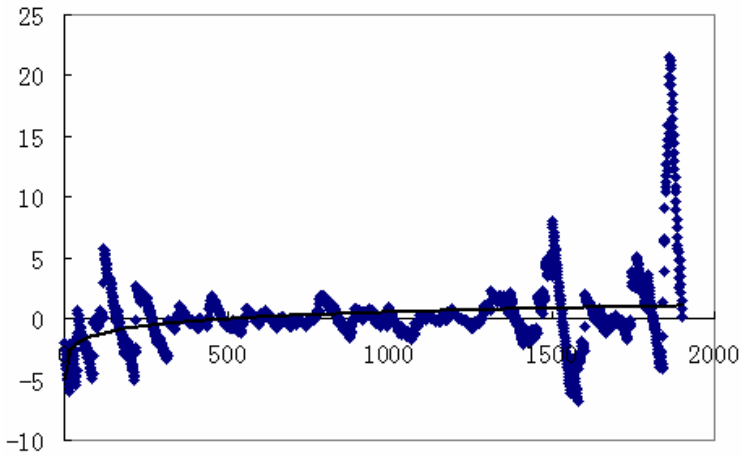

Fig. 1. Gradient of block error using best case DSS individual. X axis represents tournament and Y represents slope of best fitting line on 100 point window

On the basis of these observations, the number of DSS selections per block is limited to 100 (from 400) – with the objective of further reducing any tendency to prematurely specialize – where the principle cost is in a higher number of block selections, 1000 in this case.

3.3 Structural Removal of Introns

Introns are program pieces, which have no influence to the output, but appear to be a factor in the evolution of solutions. Moreover, two forms of introns are often distinguished: structural introns and semantic introns. Structural introns manipulate variables that are not used for the calculation of the outputs at that program position. Whereas, semantic introns manipulate variables on which the state of the program is invariant [15]. In this work, structural introns are detected using following pseudo code, initiated once evolution is complete, with the last reference to R[0] (the register *a priori* defined as the output) as the input argument.

```
markExon(reg, i)

{..for (destination in the i^th instruction != reg; i--)
   if (i = 0) exit;
   mark i^th instruction as exon
   markExon(oprand1, i-1)
   markExon(oprand2, i-1)   .... }
```

4 Experiment

The following experiments are based on 40 runs using Dynamic Page-based L-GP. Runs differ only in their choice of a random seed initializing the population. Table 2 lists the common parameter settings for all runs. The total number of records in training and test set is listed in Table 3. The method used for encouraging the identification of temporal relationships and composing the instruction set is defined as follows.

Sequencing Information. As indicated above, only the 8 basic features of each connection are used, corresponding to: Duration; Protocol; Service; normal or error status of the connection (Flag); number of data bytes from source to destination (DST); number of data bytes from destination to source (SRC); LAND (1 if connection is from/to the same host/port, 0 otherwise); and number of "wrong" fragments (WRONG). This implies that GP is required determine the temporal features of interest itself. To do so, for each 'current' connection record, $x(t)$, GP is permitted to index the previous 32 connection records relative to the current sample, modulo 4. Thus, for each of the eight basic TCP/IP features available in the KDD-99 dataset, GP may index the 8 connection records $[(t), (t-4), \ldots (t-32)]$, where the objective is to provide the label associated with sample 't'.

Table 2. Parameter Settings for Dynamic Page based Linear GP

Parameter	Setting
Population Size	125
Maximum number of pages	32 pages
Page size	8 instructions
Maximum working page size	8 instructions
Crossover probability	0.9
Mutation probability	0.5
Swap probability	0.9
Tournament size	4
Number of registers	8
Instruction type 1 probability	0.5
Instruction type 2 probability	4
Instruction type 3 probability	1
Function set	{+, -, *, /}
Terminal set	$\{0, .., 255\} \cup \{i_0, .., i_{63}\}$
RSS subset size	5000
DSS subset size	50
RSS iteration	1000
DSS iteration (6 tournaments/ iteration)	100
Wrapper function	0 if output <=0, otherwise 1
Cost function	Increment by 1 for each misclassification

Table 3. Distribution of Normal and Attacks

Connection	Training	Test
Normal	97249	60577
Attacks	396744	250424

Instruction Set. A 2-address format is employed in which provision is made for: up to 16 internal registers, up to 64 inputs (Terminal Set), 5 opcodes (Functional Set) – the fifth is retained for a reserved word denoting end of program – and an 8-bit integer field representing constants (0-255) [12]. Two mode bits toggle between one of three instruction types: opcode with internal register reference; opcode with reference to input; target register with integer constant. Extension to include further inputs or internal registers merely increases the size of the associated instruction field. The output is taken from the first internal register.

Training was performed on a Pentium III 1GHz platform with a 256M byte RAM under Windows 2000. The 40 best individuals within the last tournament are recorded and translated simplified as per Section 3.3. Note that 'best' is defined with respect to the cost function used during training, Table 2. Performance of these cases is then expressed in terms of false positive (FP) and detection rates, estimated as follows,

$$Detection\,Rate = 1 - \frac{\#False\ Negatives}{Total\ Number\ of\ Attacks} \tag{6}$$

$$False\ Positive\ Rate = \frac{\#False\ Positives}{Total\ Number\ of\ Normal\ Connections} \tag{7}$$

Figure 3 summarizes the performance of all 40 runs in terms of FP and Detection rate on both training and test data. Of the forty cases, three represented degenerate solutions (not plotted). That is to say, they basically classified everything as normal (i.e. only 20% of the training classifications would be correct) or attack (roughly 80% of the training connections would be correct). Outside of the case of the three degenerate cases, it is apparent that solution classification accuracy is consistently achieved. Table 4 makes a direct comparison between KDD-99 competition winners, verses the corresponding GP cases.

Structural removal of introns, Section 3.3, resulted in a 5:1 decrease in the average number of instructions per individual (87 to 17 instructions). With the objective of identifying what type of rules were learnt, the GP individual with best Detection Rate from Table 4 was selected for analysis. Table 5 lists the individual following removal of the structural introns.

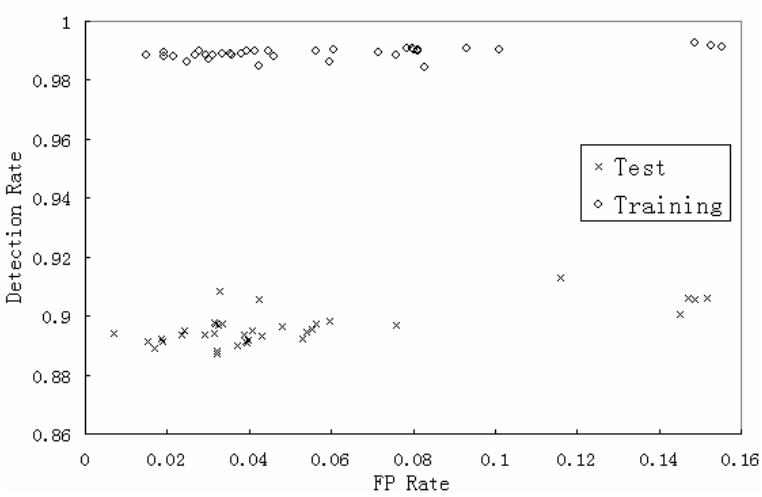

Fig. 3. FP and detection rate of 40 runs on KDD-99 test data set

Table 4. Comparison with KDD-99 winning entries

Parameter	Detection Rate	FP rate
Winning entry	0.908819	0.004472
Second place	0.915252	0.00576
Best GP – FP rate	0.894096	0.006818
Best GP – Detection rate	0.908252	0.032669

Table 5. Anatomy of Best Individual

Opcode	Destination	Source
LOD	R[0]	20
SUB	R[0]	Input[2][5]
MUL	R[0]	Input[0][1]
DIV	R[0]	Input[0][4]
SUB	R[0]	Input[2][5]
SUB	R[0]	Input[6][5]
DIV	R[0]	Input[0][4]

Table 6 summarizes performance of the individual over a sample set of the connection types in terms of connections types seen during training (24 different types) and connections types only during test (14 different types). Of particular interest here is that high classification accuracy is returned for connection types, which are both frequent and rare, where it might be assumed that only the connections with many examples might be learnt.

Table 6. Error rates on test data for top 16 attacks by individual with Best Detection Rate

Seen connection type	% Misclassified	Total Examples	Unseen connection type	% Misclassified	Total Examples
Neptune	0	58,001	Udpstorm	0	2
Portsweep	0	354	Prosstable	3.03	759
Land	0	9	Saint	5.978	736
Nmap	0	84	Mscan	8.452	1,053
Smurf	0.077	164,091	Httptunnel	15.823	158
Satan	3.552	1,633	Phf	50	2
Normal	3.267	60,577	Apache2	65.491	794

Re-expressing this individual analytically, below, indicates that the statistics of the number of bytes from the responder and the byte ratio responder-originator are utilized. This enables the individual to identify that the attacking telnet connections in the DARPA dataset are statistically different from the normal telnet connections. Moreover, not only telnet connections can be classified by this way. Such a rule never misses an attack of "Neptune", "portsweep", "land", "nmap", "udpstorm". It also provided 'good' performance on "smurf", "processtable", "normal", "satan", "saint", "mscan" and "httptunnel". For "Neptune," there are many half open tcp connections, without any data transfer. In "smurf," there are many echo replies to victim, but no echo requests from victim. In "http tunnel," the attacker defines attacks on the http protocol, which is normal, but the actual data exchange ratio makes it different from normal traffic. Currently, only [16] argued that telnet connection can be differentiated by a rule of the form discovered here. It has been suggested that attacks be formulated with such a rule in mind, [17], but without explicitly proposing using this statistic. Thus GP in this case has actually provided a unique generic rule for the detection of multiple attack types.

$$Output = \frac{\frac{(20 - Input[2][5]) \times Input[0][1]}{Input[0][4]} - Input[2][5] - Input[6][5]}{Input[0][4]}$$

where $Input[j][i]$ indexes the i^{th} input feature at temporal location $t - 4 \times j$; and 't' (= 0) is the current connection.

5 Conclusion

A Page-based Linear Genetic Programming system with DSS and RSS was implemented and tested on the KDD'99 benchmark dataset, a problem involving a training dataset of half a million patterns. To do so, a *hierarchy* of data subset selections is introduced such that GP only perceives 50 of the total training set patterns at any one

time. Moreover, such a hierarchy is designed to utilize the memory hierarchy commonly employed in computer architectures. As such the ensuing system completes each trial in approximately 15 minutes on a modest laptop-computing platform (1Ghz Pentium III, 256 Mbyte RAM) or 10 hours for 40 trials.

In addition, only the 'basic' *connection* features are employed, with GP deriving the necessary temporal features itself. Performance approaches that of data-mining solutions based on all 41 features, whilst solution transparency is also supported and verified, enabling the user to learn from the solutions provided. Note however, that the principle design interest of this work was to demonstrate that GP could be applied to data-driven learning problems on large datasets. The resulting GP classifier represents an anomaly detector, providing a binary decision boundary: normal or attack. Extensions to include the classification of different attack types would involve training additional detectors on the subset of patterns labeled as attack. The ensuing hierarchy of detectors would provide an additional (attack) class label at each level.

Future work is expected to include a dynamic cost function; with the objective of adjusting at run time the relative weighting associated with different attack types. Moreover, the function set at present is purely analytical. Of interest would be the significance of conditional statements or modular code within this problem context.

Acknowledgements. This research was partially supported by NSERC Discovery Grants of Drs. Heywood and Zincir-Heywood.

References

1. Lippmann R.P., Fried D.J., Graf I., Haines J.W., Kendall K.R., McClung D., Weber D., Webster S.E., Wyschogrod D., Cunningham R.K., Zissman M.A.: Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation. Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, 2 (2000)
2. McHugh J.: 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 and System Security. 3(4), (2000) 262–294
3. Elkan C.: Results of the KDD'99 Classifier Learning Contest. SIGKDD Explorations. ACM SIGKDD. 1(2), (2000) 63–64
4. Wenke L., Stolfo S.J., Mok K.W.: A data mining framework for building intrusion detection models. Proceedings of the 1999 IEEE Symposium on Security and Privacy (1999) 120–132
5. Pfahringer B.: Winning the KDD99 Classification Cup: Bagged Boosting. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 65–66
6. Levin I.: KDD-99 Classifier Learning Contest LLSoft's Results Overview. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 67–75
7. Vladimir M., Alexei V., Ivan S.: The MP13 Approach to the KDD'99 Classifier Learning Contest. SIGKDD Explorations. ACM SIGKDD. 1(2) (2000) 76–77
8. Gathercole C., Ross P.: Dynamic Training Subset Selection for Supervised Learning in Genetic Programming. Parallel Problem Solving from Nature III. Lecture Notes in Computer Science, Vol. 866. Springer-Verlag, Berlin (1994) 312–321

9. Cramer N.L.: A Representation for the Adaptive Generation of Simple Sequential Programs. Proceedings of the International Conference on Genetic Algorithms and Their Application (1985) 183–187
10. Nordin P.: A Compiling Genetic Programming System that Directly Manipulates the Machine Code. In: Kinnear K.E. (ed.): Advances in Genetic Programming, Chapter 14. MIT Press, Cambridge, MA (1994) 311–334
11. Huelsbergen L.: Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations. Proceedings of the 3rd Conference on Genetic Programming. Morgan Kaufmann, San Francisco, CA (1998) 158–166
12. Heywood M.I., Zincir-Heywood A.N.: Dynamic Page-Based Linear Genetic Programming. IEEE Transactions on Systems, Man and Cybernetics – PartB: Cybernetics. 32(3) (2002), 380–388
13. Koza J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
14. Hennessy J.L., Patterson D.A.: Computer Architecture: A Quantitative Approach. 3rd Edition. Morgan Kaufmann, San Francisco, CA (2002)
15. Brameier M., Banzhaf W.: A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. IEEE Transactions on Evolutionary Computation, 5(1) (2001) 17–26
16. Caberera J.B.D., Ravichandran B., Mehra R.K.: Statistical traffic modeling for network intrusion detection. Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (2000) 466–473
17. Kendall K.: A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Master Thesis. Massachusetts Institute of Technology (1998)