SAME: An Intelligent Anti-malware Extension for Android ART Virtual Machine

Konstantinos Demertzis(✉) and Lazaros Iliadis(✉)

Democritus University of Thrace, 193 Pandazidou st. 68200, Orestiada, Greece
{kdemertz,liliadis}@fmenr.duth.gr

Abstract. It is well known that cyber criminal gangs are already using advanced and especially intelligent types of Android malware, in order to overcome the out-of-band security measures. This is done in order to broaden and enhance their attacks which mainly target financial and credit foundations and their transactions. It is a fact that most applications used under the Android system are written in Java. The research described herein, proposes the development of an innovative active security system that goes beyond the limits of the existing ones. The developed system acts as an extension on the ART (Android Run Time) Virtual Machine architecture, used by the Android Lolipop 5.0 version. Its main task is the analysis and classification of the Java classes of each application. It is a flexible intelligent system with low requirements in computational resources, named Smart Anti Malware Extension (SAME). It uses the biologically inspired Biogeography-Based Optimizer (BBO) heuristic algorithm for the training of a Multi-Layer Perceptron (MLP) in order to classify the Java classes of an application as benign or malicious. SAME was run in parallel with the Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Genetic Algorithm (GA) and it has shown its validity.

Keywords: Android malware · Java Class File Analysis (JCFA) · ART virtual machine · Multi-Layer Perceptron (MLP) · Biogeography-Based Optimizer (BBO) · Bio-inspired optimization algorithms

1 Introduction

1.1 Android Malware

Advanced generations of Android malware, often appear as legitimate social networking or banking applications. They are even involved in security systems in order to enter mobile devices and steel crucial data like the Two-Factor Authentication (2FA) sent by the banks. In this way the offer the cyber criminals the chance to have access in accounts despite the existence of additional protection level. Also this type of malware is characterized by a series of upgraded characteristics allowing the attackers to change the device control between HTTP and SMS, regardless the availability of Internet connection. They can also be used for the development of portable botnets and for spying on their victims. [1].

M. Núñez et al. (Eds.): ICCCI 2015, Part II, LNCS 9330, pp. 235–245, 2015.
DOI: 10.1007/978-3-319-24306-1_23

1.2 Android Package (APK)

An Android application is usually written in Java and it is compiled by an SDK tool, together with all of the source files in an Android package (.APK file). Such a typical application includes the Android Manifest.xml file and the classes' .dex (a file that contains the full byte code that is interpreted by the Java Virtual Machine (JVM) [2].

1.3 ART JVM

According to the changes in the Android code after the release of the Lolipop 5.0 Android OS (release date 12/11/2014) the ART JVM replaced Dalvik JVM. Dalvik JVM interpreted the Android applications in "machine language" (ML) so that they can be executed by the device processing unit only during the application execution (Just-In-Time (JIT) compiler). On the other hand, ART creates and stores the interpreted ML during the installation of the application. This is done so that it can be available all the time in the operating system (Ahead-Of-Time (AOT) compiler). ART uses the same encoding with Dalvik, by keeping the .dex files as parts of the .APK files, but it replaces the .odex files (optimized .dex files) with the corresponding .ELF ones (Executable and Linkable Format) [2]. A comparison of Dalvik and ART JVM architectures can be seen in the figure 1.

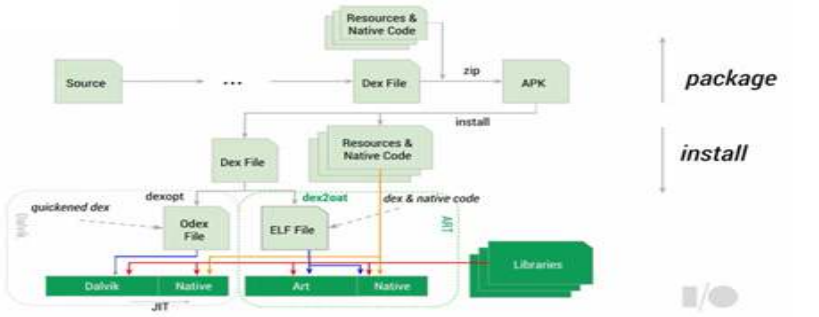

Fig. 1. A comparison of Dalvik and ART JVM architectures (photo by anandtech.com)

1.4 Java Class File Analysis (JCFA)

Generally the architecture of a Java application is described as follows: The source code Java files (.java) are compiled to byte code files (.class) which are platform independent and they can be executed by a JVM just like ART. The classes are organized in the .java files with each file containing at least one public class. The name of the file is identical to the name of the contained public class. The ART loads the classes required to execute the Java program (class loader) and then it verifies the validity of the byte code files before execution (byte code verifier) [3]. The JCFA process includes also the analysis of the classes, methods and specific characteristics included in an application.

1.5 Proposed System

This research paper, introduces advanced Artificial Intelligence (AI) methods, applied on specific parameters and data (obtained after the JCFA process) in order to perform binary classification of the classes comprising an application, in benign or malicious. More specifically, it describes the development of the SAME system, which acts as an extension of the ARTJVM. The SAME employs the Biogeography-Based Optimizer in order to train a MLP which classifies the Java classes of an application successfully in benign or malicious.

It is really important that this is achieved by consuming minimum computational resources. The proposed system enhances the Android operating system with the JCFA process.

In a second stage a comparative analysis with other timely methods like the Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Genetic Algorithm (GA) was performed with encouraging results.

1.6 Literature Review

It is a fact that a lot and significant work has been published in the literature, in applying machine learning (ML) techniques, using features derived from both static [4][5][6] and dynamic [7] analysis to identify malicious Android applications [8][9]. Yerima et al. [10] proposed a parallel machine learning approach for early detection of Android malware by utilizing several classifiers with diverse characteristics. Also, in [11], PUMA (Permission usage to detect malware in Android) detects malicious Android applications through machine-learning techniques by analyzing the extracted permissions from the application itself. Dini et al. [12] proposed a Multi-level Anomaly Detector for Android Malware (MADAM) system in order to monitors Android at the kernel-level and user-level to detect real malware infections using ML techniques to distinguish between standard behaviors and malicious ones.

On the other hand Dan Simon [13] employed the BBO algorithm on a real-world sensor selection problem for aircraft engine health estimation. Panchal et al. [14] proposed a Biogeography based Satellite Image Classification system and Lohokare et al. [15] demonstrated the performance of BBO for block motion estimation in video coding. Ovreiu et al. [16] trained a neuro-fuzzy network for classifying P wave features for the diagnosis of cardiomyopathy. In this work we employ 11 standard datasets to provide a comprehensive test bed for investigating the abilities of BBO in training MLPs. Finally Mirjalili et al. [17] proposed the use of the Bio-geography-Based Optimization (BBO) algorithm for training MLPs to reduce the problems of entrapment in local minima, convergence speed and sensitivity to initialization.

1.7 Innovation of the SAME Project

A basic innovation of the system described herein is the inclusion of a machine learning approach as an extension of the ARTJVM used by the Android OS. This join with the JCFA and the fact that the ARTJVM resolves Ahead-Of-Time all of the depend-

encies during the loading of classes, introduces Intelligence in compiler level. This fact enhances the defensive capabilities of the system significantly. It is important that the dependencies and the structural elements of an application are checked before its installation enabling the malware cases.

An important innovative part of this research is related to the choice of the independent parameters, which was done after several exhaustive tests, in order to ensure the maximum performance and generalization of the algorithm and the consumption of the minimum resources. For example it is the first time that such a system does not consider as independent parameters, the permissions required by an application for her installation and execution, unlike all existing static or dynamic malware location analysis systems so far.

Finally, it is worth mentioning that the BBO optimization algorithm (popular for engineering cases) is used for the first time to train an Artificial Neural Network (ANN) for a real information security problem.

2 Architecture of the SAME

The architectural design of the SAME introduces an additional functional level inside the ARTJVM, which analyzes the Java classes before their loading and before the execution of the Java program (class loader).

The introduction of the files in the ARTJVM, always passes from the above level, where the check for malicious classes is done. If malicious classes are detected, decisions are done depending on the accuracy of the classification. If the accuracy is high, then the decisions are done automatically, otherwise the actions are imposed by the user regarding the acceptance or rejection of the application installation. In the case that the classes are benign the installation is performed normally and the user is notified that this is a secure application. The proposed architecture is presented in the following figure 2.

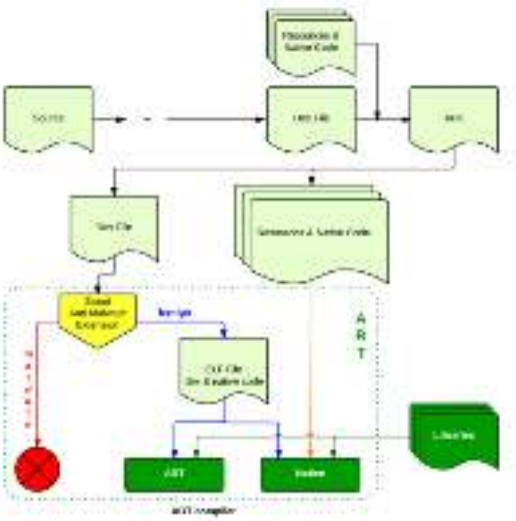

Fig. 2. The proposed architecture of the SAME

Trying a comprehensive analysis of the way the above architecture works, we clearly realize that in the proposed method, the detection and disposal of the malware is done in dead time, before its installation in the cell phone. The relative security systems presented in 1.6, analyze the operation mode of an application, based on characteristics obtained by static and dynamic analysis, by the control method of the Android Permissions and by the anomalies that can emerge in the operation of the cell phone. Of course the above process aims to spot the malware. This innovation enhances the energetic security of the Android mobile phones significantly. It also creates new perspectives in the design architecture of the operating systems, which adopt smart defense mechanisms against sophisticated attacks.

3 Dataset

3.1 Features Extraction

The dataset was constructed after using 2010 Android applications, from which 1160 were benign and were selected from the official Google Play store and 850 were malicious originating from the Drebin dataset [18][19]. The process of feature extraction was done based on the Python language, combined with the JSON technique for representing simple data structures as described in [20].

During the extract out the features process, the extracted features were related to access_flags, statistical class data and finally to methods that determine the structure of the application. The full list of the 24 features, including the class (Benign or Malicious) is presented in the following table 1.

Table 1. Extracted features

ID	Feature Name	ID	Feature Name
1	acc_abstract	13	J_class_name_slash_count
2	acc_annotation	14	J_class_name_uppercase_count
3	acc_enum	15	constant_pool_count
4	acc_final	16	entropy
5	acc_interface	17	majorversion
6	acc_public	18	method_name_digit_count
7	acc_super	19	method_name_lowercase_count
8	acc_synthetic	20	method_name_uppercase_count
9	ap_count	21	methods_count
10	J_class_name_digit_count	22	minor version
11	J_class_name_length_count	23	size
12	J_class_name_lowercase_count	24	class (Benign or Malicious)

3.2 Features Selection

As it has already been mentioned, the basic idea behind the SAME development was the use of the minimum computational resources and computational power, without making any compromise related to the performance. Thus, an effort has been made to

determine the features' vector with the well-known Principal Components Analysis (PCA) method. This was done in order for the new linear combinations to contain the biggest part of the variance of the initial information. This method was not selected because the results were not as good as expected. More specifically the mean accuracy of the algorithm was reduced by 5%.

The next step was the use of correlation analysis. In fact, we tested features' vectors that had higher correlation with the class, regardless their mutual correlation and also we tested features that had higher correlation with the class and high correlation between them. Also we tested vectors based on the cost-sensitive classification by using the cost-matrix. Additionally vectors were tested by estimating the value of each feature based on the information gain for each class (Information Gain Attribute Evaluation). Finally the optimal subset was selected based on a genetic algorithm which optimized the classification error for the training and testing data in correlation with the value of each feature [21].

3.3 Proposed Dataset

The number of parameters in the final determined dataset (after the features selection process) was reduced by 52% whereas the reduction in the total average performance of the algorithm (comparing with the performance when 24 parameters were used) was as high as 0.3% and it can be considered as insignificant. The 12 features used in the final dataset are described in the following table 2.

Table 2. Selected Features

ID	Feature Name	Interpretation
1	acc_abstract	Declared abstract; must not be instantiated.
2	acc_public	Declared public; may be accessed from outside its package
3	acc_synthetic	Declared synthetic; not present in the source code
4	J_class_name_length	The length of the class
5	J_class_name_digit_ count	The number of the digits of the class
6	J_class_name_lowercase_count	The number of the lower case of the class
7	J_class_name_uppercase_count	The number of the uppercase of the class
8	J_class_name_slash_count	Thenumber of the slash (/) characters in the name of the class
9	constant_pool_count	The constant_pool is a table of structures representing various string constants, class and interface names, field names, and other constants that are referred to within the Class File structure and its substructures. The value of the constant_pool_count item is equal to the number of entries in the constant_pool table plus one.
10	methods_count	Each value in the methods table must be a method_info structure giving a complete description of a method in this class or interface. The value of the methods_count item gives the number of method_info structures in the methods table.
11	entropy	The Entropy value was estimated for each class as degree of uncertainty, with the highest values recorded for the malicious classes.
12	class	Benign or Malicious

The value of the access_flags item is a mask of flags used to denote access permissions to and properties of this class or interface

4 Methods and Materials

4.1 MLP and Heuristic Optimization Methods

A common form of ANN is the MLP with three layers (Input, Hidden and Output). Various heuristic optimization methods have been used to train MLPs such as GA, PSO and ACO algorithms. Generally, there are three heuristic approaches used to train or optimize MLPs. First, heuristic algorithms are used to find a combination of weights and biases which provide the minimum error for a MLP. Second, heuristic algorithms are employed to find the optimal architecture for the network related to a specific case. The last approach is to use an evolutionary algorithm to tune the parameters of a gradient-based learning algorithm, such as the learning rate and momentum [22]. In the first method, the architecture does not change during the learning process. The training algorithm is required to find proper values for all connection weights and biases in order to minimize the overall error of the MLP. The most important parts of MLPs are the connection weights and biases because these parameters define the final values of output. Training an MLP involves finding optimum values for weights and biases in order to achieve desirable outputs from certain given inputs. In this study, the BBO algorithm is applied to an MLP using the first method.

4.2 Biogeography-Based Optimizer

It is a biologically inspired optimization algorithm based on biogeography, which studies the development and evolution of various species in neighboring areas called "islands". Each island is a solution to the problem whereas the species are the parameters of each solution. Each solution is characterized by variables called suitability indices (Suitability Index Variables-SIVs) [13] which define the Habitat Suitability Index-HSI [13]. The areas with the high suitability values offer the best solutions. Each solution becomes optimal by taking characteristics or species from other areas through the immigration mechanism. The algorithm is terminated either after a predefined number of iterations or after achieving a target. An essential difference with the corresponding biologically inspired optimization algorithms is that in the BBO there is no reproduction or offspring concept. There are no new solutions produced from iteration to iteration, but the existing ones are evolving. When there are changes or new solutions the parameters are notified by exchanging characteristics through the immigration [13].

4.3 BBO for Training the Proposed MLP

By introducing a .dex file to the ART the 11 features described in chapter 3.3. are exported, which comprise the independent parameters of the problem, used as independent input to the MLP. In the hidden layer 9 neurons are used, according to the following empirical function 1 [23]:

$$\left(\frac{2}{3}*\text{Inputs}\right) + \text{Outputs} = \left(\frac{2}{3}*11\right) + 2 = 9 \quad (1)$$

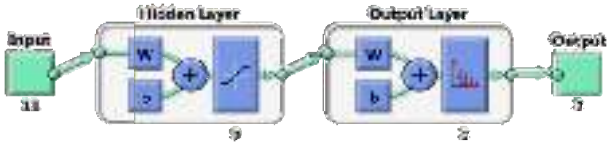

Fig. 3. Architecture of proposed MLP

The output vector of the MLP comprises of the two potential classification values benign or malicious. The architecture of the MLP is shown in the figure 3.

The proposed method starts by generating a random set of MLPs based on the defined number of habitats (BBO employs a number of search agents called habitats which are analogous to chromosomes in GAs.). Each MLP corresponds to a habitat, and each weight/bias corresponds to habitants in the habitat. After the initialization step, the MSE for each MLP is calculated by Eq. (2) [17]:

(2)

Where q is the number of training samples, m is the number of outputs, is the desired output of the ith input unit when the kth training sample is used and is the actual output of the ith input unit when the kth training sample appears in the input.

The next step is to update emigration, immigration, and mutation rates by Eqs. (3) and (4), respectively [17].

(3),

(4)

Where n is the current number of habitants, N is the allowed maximum number of habitants, which is increased by HSI (the more suitable the habitat, the higher the number of habitants), E is the maximum emigration rate, and I indicates the maximum immigration rate [17].

Afterwards, the MLPs are combined based on the emigration and immigration rates. Each MLP is then mutated based on its habitat's mutation rate.

The last step of the proposed method is elite selection, in which some of the best MLPs are saved in order to prevent them from being corrupted by the evolutionary and mutation operators in the next generation. These steps (from calculating MSE for every MLP to elitism) are iterated until satisfaction of a termination criterion. The flow chart of the proposed method can be seen in the figure 4.

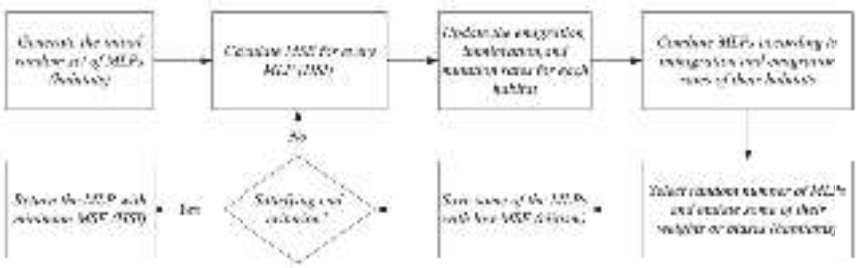

Fig. 4. Flow chart of the proposed method [17]

5 Comparative Analysis and Results

Regarding the overall efficiency of the methods, the results show that the BBO is highly suitable for training MLPs and has much better generalization performance and more accurate classification output from the other compared algorithms (the initial parameters of compared algorithms described in the [17]). The detailed accuracy by class comparison of the other bio-inspired algorithms is shown in table 3. In this case the hold out approach was used (70% Training, 15% Validation and 15% Testing).

Table 3. Comparison between BBO,PSO, ACO and GA algorithms

Classifier	ACC	MSE	TP	FP	Precision	F-measure	ROC	Class
MLP-BBO	96.7%	0.4227	0.955	0.025	0.966	0.960	0.973	Malicious
			0.975	0.045	0.967	0.971	0.973	Benign
MLP-PSO	89.4%	0.5550	0.863	0.083	0.889	0.867	0.886	Malicious
			0.917	0.137	0.898	0.908	0.886	Benign
MLP-ACO	72.1%	0.6738	0.653	0.222	0.716	0.683	0.712	Malicious
			0.778	0.347	0.724	0.750	0.712	Benign
MLP-GA	93.8%	0.4931	0.919	0.047	0.936	0.927	0.937	Malicious
			0.953	0.081	0.940	0.947	0.937	Benign

According to this comparative analysis, it appears that BBO is highly suitable for training MLPs. This algorithm successfully reduces the problem of entrapment in local minima in training MLPs, with very fast convergence rates. These improvements are accompanied by high classification rates and low test errors as well. The reasons for the better performance of BBO are as follows [17]:

- Varying values of emigration and immigration rates provide diverse information exchange behavior between habitats and consequently improve exploration.
- Over the course of generations, the HSI of all habitats are improved since habitants living in high-HSI habitats tend to migrate to the low-HSI habitats. This guarantees the convergence of BBO.
- Migration operators emphasize exploration and consequently prevent BBO from easily getting trapped in local minima.
- Different mutation rates keep habitats as diverse as possible.
- Elitism assists BBO to save and retrieve some of the best solutions, so they are never lost.

The final conclusion is that the proposed method has proven to be reliable and efficient and has outperformed at least for this dataset the other approaches.

6 Discussion – Conclusions

The Android system is very popular and thus is one of the main attack targets of the cyber criminals. This research paper presents an innovative, timely and effective AI system applied successfully in the field of Information systems' security. It is a Smart

anti-Malware Extension for Android ARTJVM, which introduces intelligence to the compiler and classifies malicious Java classes in time in order to spot the Android malwares. This is done by applying the JCFA approach and based on the effective BBO optimization algorithm, which is used to train a MLP. The most significant innovation of this methodology is that it uses embedded AI algorithms in compiler of the Android system that ensure mobile security.

Future research could involve its extension under a hybrid scheme, which will combine semi supervised methods and online learning for the trace and exploitation of hidden knowledge between the inhomogeneous data that might emerge. Also, the SAME system could be improved by optimizing further the BBO parameters. Another direction would be the use of the BBO in the training of other ANN models like the RBF NN, Cascade NN. Finally the ultimate challenge would be to test the same model in an ensemble approach.

References

1. https://blog.malwarebytes.org/
2. http://developer.android.com/
3. http://docs.oracle.com/en/java/
4. Scandariato, R., Walden, J.: Predicting vulnerable classes in an android application (2012)
5. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. In: Conference on CIS, pp. 329–333. IEEE (2010)
6. Chin, E., Felt, A., Greenwood, K., Wagner, D.: Analyzing inter-application communication in android. In: 9th Conf. on Mobile Systems, Applications, and Services, pp. 239–252. ACM (2011)
7. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: 1st ACM Workshop on SPSM, pp. 15–26. ACM (2011)
8. Glodek, W., Harang, R.R.: Permissions-based detection and analysis of mobile malware using random decision forests. In: IEEE Military Communications Conference (2013)
9. Demertzis, K., Iliadis, L.: Bio-inspired hybrid intelligent method for detecting android malware. In: Proceedings of 9th KICSS 2014, Limassol, Cyprus (2014). ISBN: 978-9963-700-84-4
10. Yerima, Y.S., Sezer, S., Muttik, I.: Android malware detection using parallel machine learning classifiers. In: Proceedings of 8th NGMAST, September 10-14, 2014
11. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G.: PUMA: permission usage to detect malware in android. In: Herrero, Á., Snášel, V., Abraham, A., Zelinka, I., Baruque, B., Quintián, H., Calvo, J.L., Sedano, J., Corchado, E. (eds.) Int. Joint Conf. CISIS'12-ICEUTE'12-SOCO'12. AISC, vol. 189, pp. 289–298. Springer, Heidelberg (2013)
12. Dini, G., Martinelli, F., Saracino, A., Sgandurra, D.: MADAM: a multi-level anomaly detector for android malware. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2012. LNCS, vol. 7531, pp. 240–253. Springer, Heidelberg (2012)
13. Simon, D.: Biogeography-based optimization. IEEE TEC **12**, 702–713 (2008)
14. Panchal, V.K., Singh, P., Kaur, N., Kundra, H.: Biogeography based Satellite Image Classification. Journal of Computer Science and Information Security **6**(2) (2009)

15. Lohokare, M.R., Pattnaik, S.S., Devi, S., Bakwad, K.M., Jadhav, D.G.: Biogeography-Based Optimization techniquefor Block-Based Motion Estimation in Video Coding CSIO 2010 (2010)
16. Ovreiu, M., Simon, D.: Biogeography-based optimization of neuro-fuzzy system parameters for diagnosis of cardiac disease. In: Genetic and Evolutionary Computation Conference (2010)
17. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Let a biogeography-based optimizer train your Multi-Layer Perceptron. Elsevier (2014). http://dx.doi.org/10.1016/j.ins.2014.01.0380020-0255/
18. Arp, D., Spreitzenbarth, M., Huebner, M., Gascon, H., Rieck, K.: Drebin: efficient and explainable detection of android malware in your pocket. In: 21st NDSS, February 2014
19. Spreitzenbarth, M., Echtler, F., Schreck, T., Freling, F.C., Hoffmann, J.: Mobile sandbox: looking deeper into android applications. In: 28th SAC 2013
20. https://github.com/ClickSecurity/data_hacking/java_classification/
21. Hall, M., Eibe, F., Holmes, G., Pfahringer, B., Reutemann, P., Witten, H.I.: The WEKA Data Mining Software: An Update. SIGKDD Explorations **11** (2009)
22. Iliadis, L.: Intelligent Information Systems and applications in risk estimation. Stamoulis Publication, Thessaloniki (2008). ISBN 978-960-6741-33-3 A
23. Heaton, J.: Introduction to Neural Networks with Java, 1st edn. (2008). ISBN: 097732060X