Detect Wi-Fi Network Attacks Using Parallel Genetic Programming

Van Canh Vu
Le Quy Don Technical University
Faculty of Information Technology
Hanoi, Vietnam
canhvuvan@yahoo.com

Tuan-Hao Hoang
Le Quy Don Technical University
Faculty of Information Technology
Hanoi, Vietnam
haohth@lqdtu.edu.vn

Abstract—**Wi-Fi network have been widely used nowadays. However, Intrusion Detection System (IDS) researches on Wi-Fi network were few and difficult since there was no common dataset between researchers on this area. Recently,** Kolias et al. [2] published a comprehensive Wi-Fi network dataset extracting from real Wi-Fi traces, which is called **the AWID dataset. Gene programming has proven effective in detecting network attacks, but the processing time is quite slow. Today, the development of GPU technology for high-speed parallel processing, the study of parallel programming solutions is essential. In this paper, we examined the Parallel Genetic Programming (Karoo GP) [13] in wireless attack detection to improve detection rates and processing time. The experiments showed that the processing time of Karoo GP was significantly improved compared to standard GP**.

Keywords— wireless attack detection; intrusion detection system; genetic programming; GP in parallel.

I. Introduction

Recently, wireless networks have been used widely. However, security experts have discovered that all Wi-Fi networks are vulnerable to hacking though there are several security protocols for Wi-Fi network such as Wired Equivalent Privacy (WEP) and Wi-Fi Protected Access 2 (WPA2) [2] to secure Wi-Fi network. Recent researches have been focused on using machine learning to detect 802.11 MAC layer attacks or impersonation attacks [6]. In this paper, our work focuses on using Genetic programming with TensorFlow to detect wireless network attacks to improve the detection rate and processing time.

Intrusion Detection System (IDS) research on Wi-Fi network was few and difficult since there was no common dataset published. Recently, Kolias et al. [2] published a comprehensive Wi-Fi network dataset, called the Aegean Wi-Fi Intrusion Dataset (AWID). AWID dataset has three different types of attacks: Impersonation, Flooding and Injection Attacks with 14 distinct attacks. Some previous works used AWID dataset to detect Wi-Fi attacks such as [1][5]. Other studies based on machine learning methods have improved the IDS effectively such as in [4][8][9]. Patrick and A. Nur [8][9] used Genetic Programming to detect de-authentication attack and Data link layer attacks. However, computation of their approach was quite heavy. In [10], Makanju et al. experimented signature-based IDSs and GP-based IDS in detecting Link Layer Attacks on Wi-Fi Networks. The results showed that GP-based attack detection systems are better classified than those based on other machine learning techniques.

In this paper, we try to examine the parallel genetic programming in wireless attack detection to improve detection rates and processing time. The paper is then conducted as follow. After the Introduction section, Section II describes some background in wireless intrusion detection and genetic programming in parallel. Section III presents and discusses some experiments and results. Conclusions and future works are shown in the last section.

II. BACKGROUND

A. Wireless Intrusion Detection Systems

Wireless local area network are subject to many types of threats and attacks such as Denial of Service (DoS), 802.11 attacks – encryption cracking, probe attacks, authentication attacks, MAC spoofing, wireless hijacking. Hackers can install rogue Wireless Access Point (rogue WAP) as a legitimate one to create backdoor into network and/or to steal sensitive data. Hackers can also send many association requests to flood WAPs and force them to reboot as malicious DoS attacks. Moreover, hackers can even use brute force attacks to decrypt sensitive data if using the standard 802.11 encryption method, WEP.

To against those attacks, Wireless Intrusion Detection Systems (WIDS) have been developed. As an intrusion detection system in general, wireless intrusion detection system gathers and analyzes data to recognize known attacks' patterns, identify intrusions and misuse and/or abnormal activities.

Jonny Miliken in [15] introduced wireless intrusion detection system structure, which contains 6 components: Theat identification, Architecture consideration, Data collection, Detection strategy, Correlation method and Evaluation as seen in Figure 1.

Threat
Architect
Collect
Detect
Correlate
Evaluate

Figure 1. WIDS Structure.

B. Genetic Programming in Parallel

Genetic programming (GP) is as an evolutionary algorithm that follows Darwin's theory evolution to generate computer programs. The first step of Genetic program system is to create the random population of individuals' program, which are tree-based structure. The second step is to evaluate the fitness of each individual in the initial random population. After that the evolution process starts to generate the new population for the next generation, where individuals in the new population are generated by evolution operators mainly crossover and mutation. The details of GP process can be seen in [19].

Thanks to GP system structure, it is easy to apply parallel computing model to GP to improve its performance. Particularly, GP is evolved as a classifier that takes a set of attributes to predict the class. The performance of GP classifier is significantly depended on the size of the training dataset. In the field of parallel Genetic Programming, many research works have been published on both CPU and GPU architectures. In most cases, two evaluation approaches: population and/or fitness are divided for simultaneously processing. H Juille and Pollack first implemented parallel GP on SIMD system for solving the trigonometric identities problem [20]. Their attempt aimed to reduce inter-processor communications. Recently, with the rapid growth of computing power in graphic cards, GPUs have been used to perform parallel GP considerably. Darren M. [21] implemented GP on graphic cards (NVidia GeForce 6400) using the data parallel approach. The proposed system was shown significant performance on benchmark problems as symbolic regressions, 11-way multiplexer and Fisher Iris dataset classification. M. Franco et al. [16] introduced a fitness parallel method using GPU with achieving the speed of up to 52x in certain datasets. Darren in [12] described a comprehensive summary of over 30 implementation efforts on computers, FPGAs, Xbox 360s, and GPUs. Augusto and Barbosa in [11] used OpenCL on GPU cores to parallel evolutionary algorithm.

C. Karoo GP

Karoo GP [17] is a Genetic Programming framework written in Object Oriented Python language. Karoo GP was developed by Kai Staats to analyze data with highly scalable vector data, multicore CPU and GPU supported through the TensorFlow library.

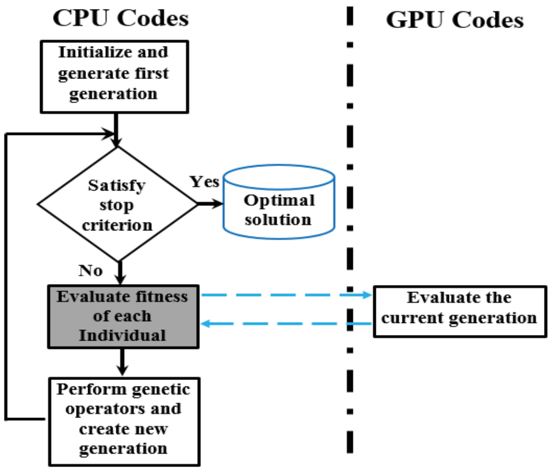
CPU Codes
GPU Codes
Initialize and generate first generation
Satisfy stop criterion
Yes
Optimal solution
No
Evaluate fitness of each Individual
Evaluate the current generation
Perform genetic operators and create new generation

Figure. 4. Overall implementation with the GPU.

Karoo_GP is included in the multicore library process, an alternative to the standard Python multiprocessing library. The multivariate expression execution generated by each tree as symbolic mathematics that was conducted by SymPy library. SymPy is quite flexible and simple to implement. In addition, TensorFlow is employed in Karoo-GP to provide the capacity of engaging massive datasets on single, multicore, and GPU architectures. With TensorFlow enabled in Karoo_GP, parallel fitness evaluation method is applied as in Figures 4 and 5.

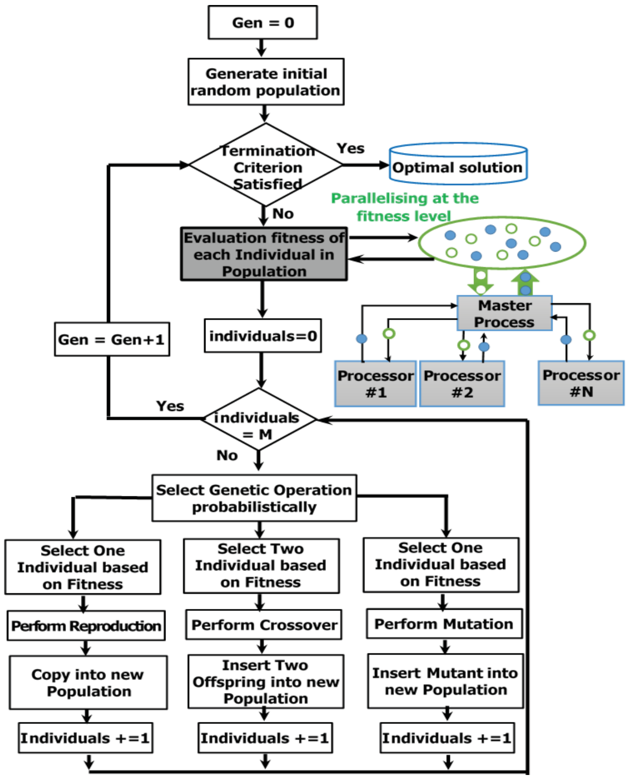
Gen = 0
Generate initial random population
Termination Criterion Satisfied
Yes
Optimal solution
No
Parallelising at the fitness level
Evaluation fitness of each Individual in Population
Master Process
Gen = Gen+1
individuals=0
Processor #1
Processor #2
Processor #N
Yes
individuals = M
No
Select Genetic Operation probabilistically
Select One Individual based on Fitness
Select Two Individual based on Fitness
Select One Individual based on Fitness
Perform Reproduction
Perform Crossover
Perform Mutation
Copy into new Population
Insert Two Offspring into new Population
Insert Mutant into new Population
Individuals +=1
Individuals +=1
Individuals +=1

Figure 5. GPU implementation of Genetic Programming

Figure 4 shows the overall GP usage on the CPU and GPU. The phases of the calculation fitness value of each individual in the population are performed on the GPU, while other tasks are performed on the CPU. In addition, the details of the proposed model are shown in Figure 5. The workflow model is described as follows:

(1) Generate the initial random population

(2) Check termination criteria

(3) Evaluate fitness value of each individual in population. This step is conducted parallel in GPU supported through TensorFlow.

(4) Select trees to be to the next generation

(5) Apply genetic operators to generate new individual to new generation

(6) Repeat from step (2) to step (5) until the termination criteria is satisfied.

III. Experiments and Results

This section presents the experimental settings in this paper and the experimental results of applying GP and Karoo_GP on AWID dataset.

A. Data settings

We used AWID dataset in [2], which contains the real trace of Wi-Fi traffic. AWIDS dataset contains two sets (AWID-CLS, AWID-ATK). AWID-CLS dataset is labeled based on 4 classes of attacks by purpose: Key cracking, Keystream, Deny of Service and Man in the Middle (or 4 classes of attacks by methodology: Passive, Injection, Flooding and Impersonation). While, AWID-ATK dataset is labeled based on the actual name of attacks: 16 classes of attacks listed in Table 1. Each of two sets has two subsets: a full subset and reduced one. In this research work, we only used the reduced subsets, which are considered the better one for doing research experiments due to their smaller size.

TABLE 1. LIST OF ATTACK TYPES IN AWID-ATK-R-TR SET

No.	Attacks
1	Amok
2	Arp
3	Beacon
4	Caffe-Latte
5	Chop-Chop
6	CTS
7	De-authentication
8	Disassociation
9	Evil-twin
10	Fragmentation
11	Hirte
12	Power-Saving
13	Probe-Request
14	Probe-Response
15	RTS
16	Normal

AWID dataset has 154 features. However, we only used 35 features of them. These features are meaningful with attacks in our experiments, as listed in Table 2.

TABLE 2. 35 FEATURES

Index	Feature Name	Description
4	frame.time epoch	Epoch Time
7	frame.time relative	Time since reference or first frame
8	frame.len	Frame length on the wire
29	radiotap.present.rxflags	RX flags
38	radiotap.mactime	MAC timestamp
74	radiotap.datarate	Data rate (Mb/s)
62	radiotap.antenna	Antenna
66	wlan.fc.type	Type
67	wlan.fc.subtype	Subtype
68	wlan.fc.ds	DS status
70	wlan.fc.retry	Retry
72	wlan.fc.moredata	More Data
73	wlan.fc.protected	Protected flag
77	wlan.da	Destination address
79	wlan.sa	Source address
80	wlan.bssid	BSS Id
82	wlan.seq	Sequence number
88	wlan.ba.bm	Block Ack Bitmap
93	wlan_mgt.fixed.capabilities. privacy	Privacy
94	wlan_mgt.fixed.capabilities. preamble	Short Preamble
98	wlan_mgt.fixed.capabilities. short slot time	Short Slot Time
104	wlan_mgt.fixed.listen ival	Listen Interva
107	wlan_mgt.fixed.timestamp	Timestamp
108	wlan_mgt.fixed.beacon	Beacon Interval
112	wlan_mgt.fixed.auth seq	Authentication SEQ
113	wlan_mgt.fixed.category code	Category code
122	wlan_mgt.tim.dtimperiod	DTIM period
125	wlan_mgt.country info.environment	Environment
126	wlan_mgt.rsn.version	RSN Version
127	wlan_mgt.rsn.gcs.type	Group Cipher Suite type
140	wlan.wep.iv	Initialization Vector
141	wlan.wep.key	Key Index
142	wlan.wep.icv	WEP ICV
144	wlan.ccmp.extiv	CCMP Ext. Initialization Vector
148	wlan.qos.ack	Ack Policy

In our experiments, we only proceeded to classify the data samples as normal or attack using GP and Karoo_GP. Based on the results obtained, we conducted a comparison between two methods. During the experiment, we used the AWIDS with 169MB and 1.048.574 samples and we trained on 20% of AWIDS dataset after that we validated on 80% of the AWIDS dataset.

B. Hardware Experiments

Our hardware configuration used to implement experiments, as follows:

- 8 CPU Intel® Xeon® Processor E3-1231 v3 (8M Cache, 3.40 GHz); 4 cores per CPU,
- Memory: 8GB (1600Mhz)
- 4 NVIDIA Corporation GF100GL [Tesla C2050]

C. Methodology settings

In classification problems where the dataset of the differential classes is quite large, the commonly used measurement is *Precision-Recall*. The method of determining *Precision* and *Recall* is depicted in Figure 7.

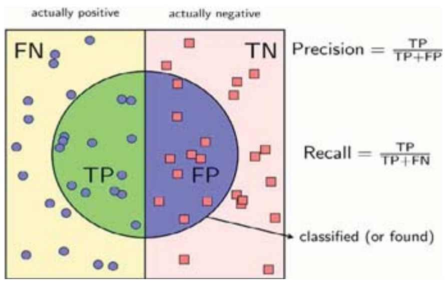

Figure 7: Precision and Recall

(i) Precision: Precision is computed as the ratio of true positive results to the total predicted positive results. High precision relates to the low false positive rate and high true positive rate. The *Precision* is defined:

$$Precision = \frac{TP}{TP + FP}$$

When *Precision* = 1, all points found are really positive, there are no false positive points in the result. However, *Precision* = 1 does not guarantee the model is good, because the model has found all the points are positive, it is not to be a good model.

(ii) *Recall (Sensitivity): Recall* is computed as the ratio of true positive results to total results. High *Recall* relates to the high true positive rate and low false negative rate. The *Recall* is defined:

$$Recall = \frac{TP}{TP + FN}$$

When *Recall = 1*, all points found are really positive, there are no false negative points in the result. However, *Recall=1* does not guarantee the model is good, because the model has found all the points are positive, it is not to be a good model.

(iii) F1-Score: F1-Score is the measure of test's accuracy F1-score is calculated based on the harmonic mean of *Precision* and *Recall*.

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

The value of *F1-Score* is in the range (0, 1]. The higher the F1-Score, the better the classifier, even in the case of *Recall* and *Precision* are equal to 1.0 (best possible), then *F1-Score* = 1. When *Recall* and precision are low, for instance recall = 0.1 and *Precision* = 0.1, then *F1-Score = 0.1*.

D. Parameter settings

In our experiments, we configured both of GP and Karoo_GP with the same parameters settings, as given in Table 3.

TABLE 3. THE PARAMETER SETTINGS

Parameter	Settings
Population size	500
Crossover	70%
Mutation	20%
Reproduction	10%
Type tree	Ramped half/half
Tree depth base	10
Tree depth max	15
Min nodes	150
Tournament	10
Function sets	+,-,*,/
Generation	50
Terminal Set	35 attributes in AWID dataset

B. Results and Discussion

Our experiment was conducted in 30 runs from 30 random seeds for the initial population creation, respectively.

The performance results for wireless attack detection using GP and Karoo_GP system are shown in Table 4 and 5.

TABLE 4. THE RESULT OF WIRELESS ATTACK DETECTION USING GP

	Precision	Recall	F1-score	Support
Normal	0.93	0.94	0.93	200712
Attack	0.785	0.78	0.78	9003
Avg/Total	0.92	0.93	0.92	209715

TABLE 5. THE RESULT OF WIRELESS ATTACK DETECTION USING KAROO_GP

	Precision	Recall	F1-score	Support
Normal	0.99	0.99	0.99	200712
Attack	0.82	0.79	0.80	9003
Avg / Total	0.98	0.98	0.98	209715

As can be seen, the results in Table 4 and Table 5 show that using K_GP to classify wireless attacks has far better results than using GP standard, the precision rate of normal patterns when using Karoo_GP is 99% while GP is 93%. For attack patterns, the Precision rate is 82% with Karoo_GP and 78.5% for GP. Therefore, Precision rate in classification when using Karoo_GP is higher than GP standard. Moreover, when using Recall measurement, Karoo_GP has better results than GP on classifying normal patterns with 99% and attack patterns with 79%; while GP just achieved 94% for normal patterns and 78% for attack patterns.

F1-Score for both normal and attack patterns classification results of Karoo_GP are better than that of standard GP. Particularly, F1-Score of Karoo_GP is 99% for normal patterns and 80% for attack patterns while its standard GP is 93% and 78% for normal and attack patterns, respectively.

In addition, the average of Precision, Recall and F1-Score of Karoo_GP are also higher than the standard GP. For Karoo_GP, the average values of Precision, Recall, and F1-Score are all 98%, while those of standard GP respectively are 92%, 93%, and 92%.

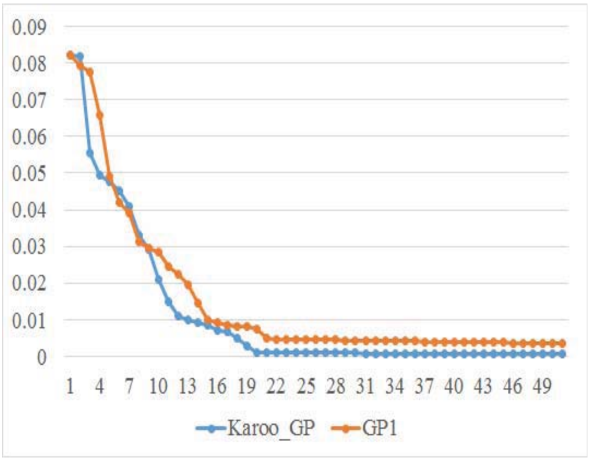

Figure 8. Fitness values of Karoo_GP and GP techniques

Figure 8 depicts the average of the best fitness in each generation. It is clearly seen that Karoo_GP converges faster than the standard GP and the performance of Karoo_GP is better than its standard GP.

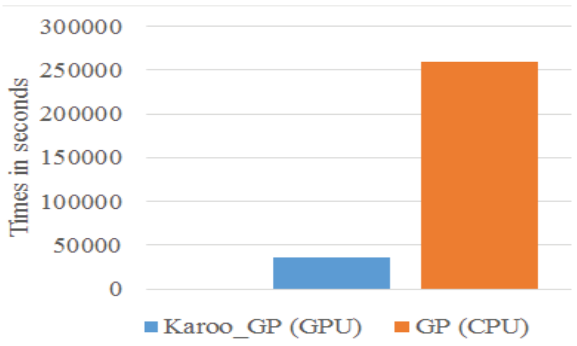

Figure 9. Compares the processing times of Karoo_GP (GPU) and GP (CPU).

In our experiments, Karoo_GP and standard GP were implemented on the same computer system. However, Karoo_GP was implemented in parallel processing with GPU configuration as described in Section II.B; while standard GP was not. Figure 9 shows that the execution time of Karoo_GP is much faster than the standard GP. Particularly, Karoo_GP processing time is about 10 hours while standard GP is about 72 hours.

IV. Conclusion and Future works

In this paper, we presented a genetic programming technique in parallel to classify data patterns for wireless attacks detection. We conducted experiments to compare the performance of Karoo_GP and standard GP on the same configuration of the computer system, the same parameters setting and datasets. However, Karoo_GP was implemented in a parallel processing mechanism with the high-speed GPU configuration, while the standard GP only processed on multi-core CPUs. The results showed that the execution time of Karoo_GP is much faster than its standard GP. In addition, Precision, Recall, and F1-Score rate of Karoo_GP are also higher than conventional standard GP, the average value is 98% for Karoo_GP and 92% for standard GP.

In the near future, we will conduct experiments for multiple classifications on Parallel Genetic Programming. In addition, we will implement Parallel TAG3P (Tree Adjoining Grammar-Guided Genetic Programming) [18] to get the better performance on processing and accuracy in data classification.

References

[1]. Bandar Alotaibi; Khaled Elleithy: “A majority voting technique for wireless intrusion detection systems,” 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, NY, USA, pp.1-6

[2]. Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, Stefanos Gritzalis: “Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset,” IEEE Communications Surveys and Tutorials, Volume 18 Issue, pp. 184-208. IEEE (2015)

[3]. Sabhnani, M. and Serpen, G.: “Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context,” In Proceedings of the International Conference on Machine Learning: Models, Technologies, and Applications. pp. 209-215. (2003)

[4]. Sommer, R., Paxson, V.: “Outside the closed world: On using machine learning for Network Intrusion Detection,” Security and Privacy, 2010 IEEE Symposium on (2010)

[5]. Usha, M., Kavitha, P.: “Anomaly based intrusion detection for 802.11 networks with optimal features using SVM classifier,” Springer Wireless Networks, the Journal of Mobile Comm., Computation and Information, 22, pp. 1–16. Springer (2016)

[6]. Y. Sheng, K. Tan, G. Chen, D. Kotz, A. Campbell: “Detecting 802.11 MAC Layer Spoofing Using Received Signal Strength,” INFOCOM 2008. The 27th Conference on Computer Communications. IEEE

[7]. Cisco Visual Networking Index: “Global Mobile Data Traffic Forecast Update” 2016–2021 White Paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[8]. Patrick LaRoche, A. Nur Zincir-Heywood, “802.11 De-authentication Attack Detection Using Genetic Programming”, Genetic Programming: 9th European Conference, EuroGP 2006, Budapest, Hungary, April 10-12, 2006. Proceedings (pp.1-12)

[9]. Patrick LaRoche, A. Nur Zincir-Heywood, “Genetic Programming Based Wi-Fi Data Link Layer Attack Detection,” Communication Networks and Services Research Conference, 2006. CNSR 2006. Proceedings of the 4th Annual Communication Networks and Services Research Conference, Washington DC, USA, pp.285-292. IEEE Computer Society, Los Alamitos (2006).

[10]. Makanju, A.; LaRoche, P.; Zincir-Heywood, A.N., “A comparison between signature and GP-based IDSs for link layer attacks on Wi-Fi networks,” Proceedings of the 2007 IEEE Symposium on Computation Intelligence In Security and Defense Applications, CISDA 2007, pp.213-219, 1-5 April 2007

[11]. Douglas A. Augusto and Helio J.C. Barbosa: “Accelerated parallel genetic programming tree evaluation with OpenCL”. Journal of Parallel and Distributed Computing, Volume 73, Issue 1 (2013), pp.86–100.

[12]. Darren M. Chi1y: “Fast parallel genetic programming: multi-core CPU versus many-core GPU,” Soft Computing volume 16, Issue 10 (2012), pp.1795–1814.

[13]. Kai Staats, Karoo GP User Guide Genetic Programming in Python, www.kaistaats.com/research/

[14]. Douglas A. Augusto, Heder S. Bernardino and Helio J.C. Barbosa: “Genetic programming: Chapter 5. Parallel Genetic Programming on Graphics Processing Units”, pp.95-114, http://dx.doi.org/10.5772/48364

[15]. Jonny Miliken, The state of the Art Intrusion Prevention and Detection: Introduction to Wireless Intrusion Detection System, pp.335-360.

[16]. M. A. Franco, N. Krasnogor, and J. Bacardit: “Speeding up the evaluation of evolutionary learning systems using GPGPUs,” In Proceedings of the 12th annual conference on Genetic and evolutionary computation, GECCO10, pp.1039–1046, New York, USA, 2010. ACM.

[17]. Kai Staats, Edward Pantridge, Marco Cavaglia, 2017, TensorFlow Enabled Genetic Programming, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Companion, ACM 2017, Berlin, Germany , pp. 1872-1879

[18]. Hoai N.X., McKay R.I.B., Essam D., Hoang Tuan-Hao: "Genetic Transposition in Tree-Adjoining Grammar Guided Genetic Programming: The Duplication Operator," European Conference on Genetic Programming, EuroGP 2005. Lecture Notes in Computer Science, vol 3447, pp.108-119 Springer, Berlin, Heidelberg.

[19]. John R. Koza: "Genetic programming - on the programming of computers by means of natural selection". Complex adaptive systems, MIT Press 1993, ISBN 978-0-262-11170-6, pp.I-XVIII, 1-419

[20]. H. Juillé & J. B. Pollack, Parallel Genetic Programming and fine-grained SIMD architecture" in Working Notes for the AAAI Symp. Genetic Programming, E. V. Siegel and J. R. Koza, Eds. Cambridge, MA: MIT, Nov. 10-12, 1995, AAAI, pp.31-37.