Evolutionary Memetic Models for Malware Intrusion Detection: A Comparative Quest for Computational Solution and Convergence

I. P. Okobah
Department of Computer Science Education,
Federal College of Education Technical
Asaba, Delta State,
Nigeria

A. A. Ojugo
Department of Computer Science, Federal
University of Petroleum Resources
Effurun, Delta State,
Nigeria

ABSTRACT
Data security is now a pertinent issue with advent of the Internet. Methods like cryptography, firewalls and gateways used to prevent attacks on data are becoming unsuccessful. Thus, the need for Intrusion Detection System to enhance security efforts. Varying machine learning models are implemented for rule-based IDS using DARPA dataset to train and generate rules for classification via support-confidence framework and a common fitness function to judge quality of each rule. This will help detect network anomalies, new attack types via rules and allow their addition into knowledgebase. Study presents results of the various stochastic models used with an aim to improve data security and integrity for networked resources.

Keywords
Intrusion, evolutionary, forensic, data security, adversaries, hackers.

1 INTRODUCTION
TODAY's society demands a great dependence on digitally transmitted data as people seek to become more effective and efficient in their daily dealing. Though, a current menace ameliorating this need is also the drastic increase of hackers, who illegally gain access to authentic data. Many systems and studies have ensued in a bid to find means and methods that will help desuade these adversaries from such act. This has consequently, led to advances in digital forensics using many existing data mining tools. Also, other means have been the implementation of security tools such as firewalls, application gateways etc – all of which seeks to ensure data integrity, non-repudiation and privacy. This has become and is now a herculean task as new direction now focuses on intrusion detection systems to help administrators monitor network data-traffic and thus, help them identify resource misuse, unauthorized use and abuse on networks (Ojugo et al, 2012; Ojugo et al, 2016).

Threats are initiated externally or internally (Gong et al, 2005; Kandeeban and Rajesh, 2007) – to result in 2-types of intruders: external (with no authorized access to resources; But, attacks via penetration means) and internal intruder (with authorized network access to resources). To ensure data integrity, firewall's demerit is its faulty packet filtering. Application gateway's demerit is its cost and bottleneck caused as it slows down the network. Thus, IDS aims to bridge existing network security technologies. Intrusions are a set of action that attempts to compromise integrity, confidentiality, privacy and availability of network resources. The intruder (or adversary as often referred to) is any user or group of users who initiates such intrusive action (Olusegun et al, 2010). Thus, IDS is designed to generate alert as it observes potentially malicious and abusive traffic (Kurose and Ross, 2010). It monitors data packets from network connections and determines if it is an intrusive activity or not. If an intrusive action is detected, the IDS performs one of these: (a) logs a message into the system audit file to be later analyzed by a security expert, (b) emails an alert to a network administrator, and (c) ends the adversary's connection (that is as provisioned under Intrusion Prevention System) amongst other functions.

1.1 Intrusion Detection Systems
Diaz-Gomez and Hougen (2005) IDS has 3-main parts namely: (a) sensors/network probes which tracks data traffic, system behavior and log files by translating data into events usable as the IDS monitors and taps in to access all network connections, (b) analysis console which takes sensor output as input in form of network connections, analyzes it for intrusive acts (as critical component to decide whether or not, a connection is intrusive), and (c) policy control which generates reactions based on analysis' outcome. If analysis console flags a connection as intrusion, control performs several actions depending on policies, set by the network administrator. Such actions include simple logout of a particular connection, alerting the administrator via e-mail etc. It also handles action(s) to be taken when an intrusion is detected as in Figure 1.

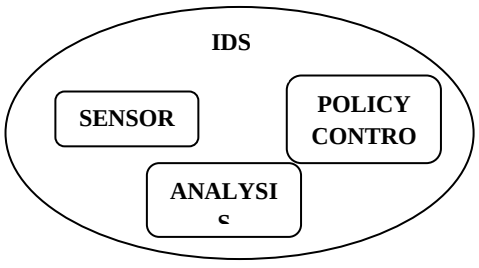

Figure 1: Framework of an IDS

Ojugo et al (2012b) IDSs are basically classified into:

1.1.1 *Anomaly-IDS*
Creates profile statistics of unusual packet stream of Internet Control Message Protocol, ping sweeps and port scan's sudden exponential growth. They do not rely on previous knowledge to detect new, undocumented attacks; but, it is a herculean task to distinguish normal from statistically unusual traffic (Kurose et.al, 2010).

1.1.2 *Signature/Rule-Based*

Maintains an attack database of signatures using varying rule set(s) or heuristics that include a list of feats about a packet. They are normally created by security experts so that the IDS can sniff each packet passing the system, compare them against its database, and if packet(s) matches, it generates an alert; else, it proceeds (Kurose et.al, 2010). Its demerits: (a) it requires previous knowledge of known attacks to generate accurate rules, (b) It's completely blind to new unrecorded attack, (c) false negative classification resulting from feats not in the rulesand not an attack, (d) addition of new rules enlarges knowledgebase and packets compared against this may overwhelm the IDS processing so that it fails to detect attacks and network anomalies.

1.1.3 *Network IDS*

They identify intrusive network connection acts via monitoring traffic through network devices.

1.1.4 *Host-IDS*

Monitors file and process activities that are related to a software environment, as associated with a specific host and listens to network traffic to identify attacks to such host – as data from a host is used to detect signs of intrusion as many packets enters and/or exits a host machine (Li, 2004).

1.2 Intrusive Actions and Malware

Intrusive actions by hacker are achieved via use of malware. It is a common technique used by adversaries (Ojugo et al, 2016). A computer malware (or virus) is a malicious program that modifies a host machine by attaching its code and alters behaviour of other files. As it infects, it also modifies itself to include better and possibly, an evolved copy of the virus (Daodu and Jebril, 2008; Dawkins, 1989; Zakorzhevsky, 2011). Malware self-replicates its codes onto a machine without the user's consent, and spreads by attaching a copy of itself to some part of program file. It attacks system resources and is designed to deliver a payload that aims to corrupt program, delete files, reformat disks, crash network, destroy critical data or embark on other damage to the host machine (Szor, 2005).

Viruses have 3-modules namely: infect, trigger and payload. Infect show its mechanism to modify its host and contain copies of it. Trigger details when and how to deliver its payload; while the payload details damage done. Trigger and payload are optional (Desai, 2008). Figure 2a shows its structure of the virus English-like pseudo-code; while Figure 2b shows its internal workings (Ye et al, 2008; Ojugo et al, 2016).

```
Def Virus():
    Infect()
    If Trigger() is TRUE then
        Payload is delivered()
```

Figure 2a: Virus Pseudo-code

```
Def Infect():
    Repeat M times()
    Target = Select_target()
      If no target() THEN
            Return
  Infect code(target)
```

Figure 2b: Infect Pseudo-code

Mishra (2003), Orr (2006) and Ojugo et al (2016) Viruses (or malware) are classified into:

1.2.1 *Simple virus*

Replicates itself if launched. It gains control of the system, attaches copy of itself to another program as it spreads. After which, it transfers control back to host program. It is easily detected via search/scan for a defined sequence of bytes, known as a signature to find the virus.

1.2.2 *Encrypted viruses*

Scramble their signature so as to make it unrecognizable during execution. Its decryption routine transfers control to its decrypted virus body so that each time it infects a new program, it makes copy of both the decrypted body and its related decryption routine. It then encrypts a copy and attaches both to a target system. It uses an encryption key to encrypt its body. As the key changes, it scrambles its body so that virus appears different from one infection to another. Such virus is difficult to detect via signature. Thus, antivirus must scan for a constant decryption routine instead.

1.2.3 *Polymorphics*

Consists of a scrambled body, mutation engine and decryption routine. The decryption routine gains control to decrypt both its body and mutation engine. It then transfers control to the scrambled body to locate a new file to infect. It copies its body and mutation engine into RAM, and invokes its mutation engine to randomly generate new decryption routine to decrypt its body with little or no semblance to the previous routine. It then appends this newly encrypted body, a mutation engine and decryption routine to the newly infected file. Thus, its encrypted body and decryption routine, varies from one infection to another. With no fixed signature and decryption routine, no two infections is alike.

1.2.4 *Metamorphics*

Avoids detection by rewriting completely, its code each time it infects a new file. Its engine achieves code obfuscation and metamorphism, which in most cases – is 90% of its assembly language codes.

1.3 The Metamorphic Malware

As above, metamorphic viruses employs a mechanism that helps them to change their code structure and appearance; while, keeping its original functionality. It achieves this via code obfuscation methods as in fig 3. Its engine reads in a virus executable, locates code to be transformed using its locate_own_code module. Each engine has its transformation rule that defines how a particular opcode or a sequence of opcodes is to be transformed. Decode module extracts these rules by disassembling. Analyze module analyzes current copy of virus and determines what transforms must be applied to generate the next morphed copy. Mutate module performs the actual transformations by replacing an instruction (set) with the other its equivalent code; While, Attach module attaches the mutated or transformed copy to a host (Cohen, 1987; Desai, 2008; Orr, 2007 and Sung et al, 2004).

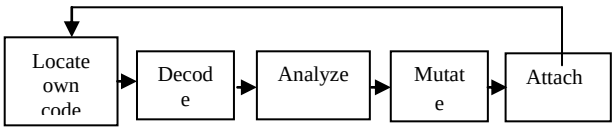

Figure 3: Distinct Signature of Metamorphic

Venkatesan (2008) note that a typical metamorphic engine may consist of: (a) internal disassemble to disassemble binary codes, (b) a shrinker replaces two or more codes with its single equivalent, (c) an expander replaces an instruction with many codes that performs same action, (d) a swapper reorders codes by swapping two/more unrelated codes, (e) a relocator assigns and relocates relative references such as jumps and calls, (f) a garbager (constructor) inserts whitespaces (do-nothing codes) to the program, and (g) cleaner (destructor)

undoes the actions of a garbager by removing whitespaces/do-nothing instructions (Desai, 2008; Konstatinou, 2008).

Feats of an effective metamorphic engine includes: (i) must be able to handle any assembly language opcode, (ii) shrinker and swapper must be able to process more than one instruction concurrently, (iii) garbager is used moderately, not to affect actual instructions, and (iv) swapper analyzes each instruction so as not to affect next instructions' execution (Orr, 2007; Sung et al, 2004; Walenstein et al, 2007).

1.4 Metamorphic Code Obfuscation Methods

Metamorphic engine uses code obfuscation to yield morphed copies of original program. Obfuscated code is more difficult to understand and can generate different looking copies of a parent file as it operates on both control flow and data section of a program (Wong, 2006). Code obfuscation is achieved via (Borello and Me, 2008; Desai, 2008 and Avcock, 2006):

a. Register Usage Exchange/Renaming – modifies the register data of an instruction without changing the codes itself, which remain constant across all morphed copies. Thus, only the operands changes.

b. Dead Code inserts do-nothing (whitespace) codes that do not affect execution via a block or single instruction so as to change codes' appearance while retaining functionality.

c. Subroutine Permutation aims to reorder subroutines so that a program of many subroutines can generate (n-1)! varied routine permutations, whose addition will not affect its functionality as this is not important for its execution.

d. Equivalent Code Substitution replaces instruction with its equivalent instruction (or blocks). A general task can be achieved in different ways. Same feat is used in equivalent code substitution.

e. Transposition/Permutation – modifies program execution order only if there is no dependency amongst instructions.

f. Code Reorder inserts unconditional and conditional branch after each instruction (or block), and defines branching instructions to be permuted so as to change the programs' control-flow. Conditional branch is always preceded by a test instruction which always forces the execution of the branching instruction.

g. Subroutine Inline/Outline is similar to dead code insertion in that subroutine call are replaced with its equivalent code as Inline inserts arbitrary dead code in a program; while outline converts block of code into subroutine and replace the block with a call to the subroutine. It essentially does not preserve any logical code grouping.

1.5 Stochastic IDS Model

Axelson et al (2004) built IDS with unsupervised Kohonen map. They proposed two enhancements that solved issue of high value of false positive rate with Performance-Based Ranking. It works by deleting an input from the dataset and comparing the result before and after the deletion using KDD dataset. To counter low detection and high false alarm rates, they proposed interactive IDS based on Bayesian statistics combined with a visualization component. Gong et al (2005) Generated rules for optimality, to detect network anomalies. Each agent monitors a network feat of audit data, and GA was used to find a set of agents that collectively detect network behavior anomalies. Its merit is application of many small autonomous agents. Its demerits are its communication defect and time consuming training if agents are not properly initialized using 6-connection feats.

Kayacık et al (2006) focused on behavioral models of known attacks to help experts identify similarities between attacks. Self-Organizing Map is used to model relationship between known attacks and U-Matrix representation in 2D topological map of known attacks evaluated on KDD dataset.

Results show attacks with similar behavior patterns were placed together on map.

Zanero et al (2008) proposed a novel architecture for the network-based anomaly detection system using unsupervised learning to describe how pattern recognition feats of a Self-Organizing Map are used on payload of TCP data-packets via ULISSE, 2-tier architecture unsupervised learning to perform intrusion and anomaly detection. Kandeeban et al (2010) had its best rule of fitness close to 1 with 97% correctly detected attacks and 0.69% of normal connections incorrectly classified as attacks. Correlation between correct attack detection and false positive rate – so that the more attacks are correctly detected, more normal connections are incorrectly classified. A major difficulty was in its establishing of its threshold value that may lead to detect novel unknown attacks.

Vollmer et al (2011) combined approaches to create rules and retrieved packets for training. Rather than use audit data logs, packets were originated from network traffic identified by IDS as being anomalous. It yielded set of optimal rules for anomalous-instance previously detected to bridge rule and anomaly IDS. Its fitness function was defined as 3-part function and demonstrated on anomalous ICMP network packets (input) and Snort rules (output). Output rules are sorted according to fitness value and duplicates removed. Out of 33,804 test packets 3 produced false positives.

Ojugo et al (2012b) Applied GA to rule based IDS using 7-feats from DARPA. Each agent was monitored a network parameter from audit data with GA used to find set of agents that collectively detect network behavior. It generated local optimum rule(s) set with 97% correctly detected attacks and 2.39% connections were incorrectly classified as attacks. A correlation between correct detection and false positive rate notes that the more attacks are correctly detected, the more normal connections are incorrectly classified as attacks.

2 MATERIALS AND METHODS

2.1 Dataset

The researcher for the purpose of this study will adopt the Win95/Zperm as implemented in on the TCP network. The DARPA 1990 dataset parameter is used as below.

2.2 Common Fitness Function

Each candidate is a solution in space with attributes over a range as encoded via integer decimal with 57-genes. IPs are HEX-coded for simplicity and quantitative representation (Ojugo et al, 2012b). Candidates are randomly initialized for selection via fitness function (weighted sum model) to indicate significance of each feature. Seven network connection feats were used (see table 1), and rules are as thus:

If {condition} then {act} *listed as:*

if {connection exists thus}:

(source IP; destination IP; source port; destination port, time)

then {stop the connection}

Table 1: Individual Representation of a Rule

Feats	Format	No. of Genes
Duration	H:M:S	3
Protocol	Numeric	1
Source Port	Numeric	1
Destination Port	Numeric	1
Source IP	a.b.c.d	4
Destination IP	a.b.c.d	4
Attack Name and type	String	1

If a connection request (source IP, destination IP, source port, destination port, and connection time) exists, then stop connection establishment. Thus, such IP is recognized as blacklisted by IDS and the service request initiated from it, is rejected. All rules are tested on historical connections, to filter new connections and find suspicious traffics. DARPA dataset 1999 is used in training and differentiates normal from intrusive connections as it contains analyzed logs with apriority attack connections. Source IP originates intrusion; while Destination IP is target whose port shows running apps. Dataset contains 6-feats as some IP are more likely targets than others and analysis prior to its use notes that normal connection contains no attack name. Agents are **rules** with 7-feats, encoded via decimal, fixed length vector. Each If-Then clause has a condition and outcome part (see table 1) of 6-feats connected via logical AND to form **condition**; while attack name is **outcome** to show network ***classification*** (in training), or connection (at detection) if a rule is matched (Ojugo et al, 2012b). Example of a port scan attack is as follows:

IF (time="0:0:1" **AND** protocol="telnet" **AND** source port=89 **AND** destination port=23 **AND** source IP="9.9.9.9" **AND** destination IP="172.16.12.50") THEN (attack="port-scan"). If telnet/port-scan is represented by integers 1 and 2:

{0, 0, 1, 2, 18982, 79, 9, 9, 9, 9, 172, 16, 012, 50, 1}

Rules are evaluated via fitness function to determine its fit and goodness with detecting attacks. A good rule correctly classifies an attack; Else, it is bad. The support-confidence fitness is used. Changing weights w1/w2 detect intrusions (with w1=0 and w2=1); while to precisely classify intrusions and detect behaviour anomalies (w1=1 and w2=0). Support-confidence fitness is as thus:

If w1 = 0.2, w2 = 0.8, N = 10, |A| = 2, |A and B| = 1.

Support = | A & B | / N = 1 / 10 = 0.1:

Confidence = |A & B | / A = 1 /2 = 0.5

Fitness = w1 * support + w2 * confidence =0.42

Table 2: Fitness Value Framework

Time	*Prot*	*Sourc Port*	*D Port*	*Source IP*	*Dest. IP*	*Attac k*
0.0.11	ftp	1892	21	192.168.1.30	192.168.1.20	-
0 .0 .0	Smtp	1900	25	192.168.1.30	192.168.1.20	-
0.0.2	**Rsh**	**1023**	**102**	**192.168.1.30**	**192.168.1.20**	**Rcp**
0.0.23	telnet	1906	23	192.168.1.30	192.168.1.20	Guess
0.0.14	rlogin	1022	513	192.168.1.30	192.168.1.20	rlogin
0.0.2	**Rsh**	**1022**	**102**	**192.168.1.30**	**192.168.1.20**	**Rsh**
0.0. 15	ftp	4354	21	192.168.1.40	192.168.1.20	-

Table 2 is audit data with sample chromosomes as rules that identifies attack. Chromosomes match are seen in lines 3 and 6, to match Rsh attack type.

2.3 Purpose of Study

The study aims to: (a) deploy rule-based IDS with malware data, and (b) test generated rules on exsiting IDS in a bid to detect the malware and its variants within. These will speed up the process of rule generation to counters new attacks, and proffer better security and potentially reduce and/or free experts of rule creation and allows LAN administrators to generate customized rules for specific attacks faced at that level. These, will hopefully detect new attacks.

2.4 Rationale for the Study

Metamorphics transform its codes as they propagate to avoid detection by using obfuscation methods to alter its behaviour when it detects its execution within virtual machine (sandbox) as means to challenge a deeper analysis (Lakhotia et al, 2004). Virus writer use weaknesses of AVs, as limited to static and dynamic analysis, and attacks these: (a) data flow, (b) control flow graph generations, (c) procedure abstract, (d) property verification, and (e) disassembly – all means to counter scans, to identify such metamorphic viruses (Konstantinou, 2008). To mutate its code generation, metamorphics analyzes its codes and re-evaluates the variant-codes generated (as complexity of transformation in the previous generation has direct impact on its current state, how a virus analyzes and transforms code in current generation). Thus, the use of code conversion algorithm that helps them detect their own obfuscation and reordering (Allenotor 2016; Ojugo et al, 2016).

3 EXPERIMENTAL EVOLUTIONARY FRAMEWORKS

Various search methods are used to find such tasks solution such as depth search, breadth search, greedy search, iterative deepening, steepest descent, etc. Some search maximizes an objective function, must be feasible (achievable) and optimal (close to best in space). CSTs with dynamic feats make such search tedious and inexplicable to resolve – so that other means are devised to resolve such task. These have yielded in optimization models that search for optimal solution(s), chosen from a set of space that relates data-input with uncontrollable parameters and feats in system, modeled to satisfy all constraints and yield output via a mathematical structure and statistical pattern analysis to yield a new discipline termed Machine learning or soft-computing (Dos Passos, Ojugo et al, 2013b).

Machine learning is a branch of artificial intelligence (AI) deals with design of models that evolves its behaviour based on empirical (sensors and databases) data – dedicated to resolve tasks via optimization. It exploits numeric data and explores human knowledge via statistical pattern analysis, mathematical models and symbolic reasoning (Ojugo et al, 2012a) – taking advantage to capture data feats of interest, and its underlying probabilities to illustrate relationships in observed values and learns to recognize complex patterns in dataset to make intelligent decisions (Ojugo et al, 2013a).

It aims at a model that when imprecision, partial truth, uncertainty and noise is applied to its input, guarantees high quality output void of over fitting. It has led to evolutionary models that are capable of quantitative data processing to ensure qualitative knowledge and experience. Inspired by behavioural patterns in biological population and evolution laws, its tuning explores 3-dynamic feats: (a) adaptation yield agents void of local minima and high-diverse random factors introduced to slow convergence, balance exploitation versus exploration so that learning feats of change, biases its solution accordingly, (b) robustness estimates a model's effectiveness, and (c) flexible decision as uncertainty feats can impact a model's future state in forecasts while focusing on its goal and ease of blackbox integration (Ojugo, 2013b).

3.1 Genetic Algorithm Trained Neural Network (GANN)

GA as inspired by Darwinian evolution consists of a dataset chosen for natural selection with potential solutions. Individuals with genes close to its optimal solution, is fit as determined by fitness function (Perez and Marwala, 2011). Based on laws of selection, GA generates better rules via 4-operators: initialization, selection, crossover and mutation. Cultural GA is one of the many variants of GA with 4-beliefs thus: (a) Normative – specific range of values to which an individual is bound, (b) Domain has information about task, (c) Temporal has information about the search space available, and (d) spatial has topographical data about the task with time as a specific feat. In addition, CGA has an influence function that ensures that individuals (altered or not) conforms to a pool that does not violate its belief space and reduces number of possible individuals generated till an optimum is found (Reynolds, 1994; Hassan and Crossley, 2004). GA's strength is in parallel traversing with solutions from randomly generated initial pool continuously evaluated via its fitness function (Diaz-Gomez and Hougen, 2005).

For hybrid GANN, ANN initializes model via its fitness function to select new pool for crossover/mutation as:

3.1.1 *Crossover* – adopts tournament selection (to maintain diversity) in chromosomes randomly chosen. With new offspring generated every iteration, a lesser number is chosen and continues till one is chosen as parents. The goal is not to create best rule (global optimum), but set of rules good enough to detect intrusion (many local maxima). Model chooses two-random cross-over points in chromosome (see table 3) between the parents, to yield two new children in lines 3 and 4 respectively.

3.1.2 *Mutation:* Each gene chromosome may (or not) change depending on probability of mutation rate. Mutation improves population diversity needed.

Algorithm for Generation of ruleset

Input: Audit data, generations and population size.

Randomly initialize the created chromosome population.

Set w1 = 0.2, w2 = 0.8, MaxGenerations = 400 (epoch)

Set N = total number of record in training set

Set generationCounter = 0

For each chromosome in population: Set A = 0, AB = 0

For Each record in training set

If record matches chromosome

AB = AB + 1 //AB++

End If

If record matches only condition part

A = A+1 //A++

End if

End For Each record

End For Each Chromosome

Select 30-best fitted chromosome into new pool

For each chromosome in new pool/population

Apply Crossover as thus:

a. Randomly select 3-chromosomes from pool

b. Return best 2-chromosome based on fitness value

c. Apply Crossover |Select best chromosome to be parent

Apply Mutation to new offspring as thus

Set mutation threshold (between 0 and 1)

For each network attribute in chromosome

Generate a random number between 0 and 1

If random number > mutation threshold then

Generate random value with respect to data feats

Set chromosome with generated attribute value

End if: End For Each

Place newly created chromosome into population

End For each

Kill old pool, new pool now current pool

Increment generationCounter by 1

If generationCounter<MaxGeneration then goto 5

Else goto: Crossover / Mutation

End

Table 3: The 2-point Crossover for ANNGA

Time	*Prot*	*S Port*	*D Port*	*Source IP*	*Destination IP*	*Attack*
0.-1.-1	Rsh	-1	1021	192.168.-1.-1	192.168.0.-1	Rsh
0.0.5	telnet	2020	23	9.9.9.9	172.16.12.50	Port scan
				Crossover Points		
0.-1.-1	Rsh	-1	1021	192.9.9.9	172.16.112.-1	Rsh
0.0.5	telnet	2020	23	9.168.-1.-1	192.168.0.50	Port scan

The generated rules are used to evaluate the remaining dataset, and the aim of testing is to gather information of how well the rules created, can detect attacks. Two methods are used for testing namely: (a) use existing rules in rule-based IDS, and (b) build tailored rule IDS. The proposed design

requires tailored rules created from traffic and fed back for detection. The rest part of the DARPA dataset are used as incoming connection to see if generated rules can distinguish between normal and intrusive connections.

3.2 Simulated Annealing Trained Neural Network (SANN)

SA as inspired by annealing, to strengthen glass and crystals – so that it is heated until it liquefies. Then slowly cool so that as molecules settles into low energy states, it tracks and alters an individual's state, constantly evaluating its energy via its energy function. Its optimal point is found via a series of Markov chain under different thermodynamic state (Perez and Marwala, 2012). Its neighbouring state is determined by randomly changing an individual's current state via a neighbourhood function. If state of lower energy is found, individuals move to it. If neighbourhood has higher energy, individuals move to that state only if an acceptance probability condition is met. If not met, individual remains at current state (Kitandis and Bras, 1980).

Acceptance probability is difference in energies between current and neighbouring states. Temperature is initialized as high, so that individuals incline towards higher energy state – allowing individuals to explore a greater portion of space, preventing its being trapped in local optimum. As model progresses – temperature reduces with cooling and individuals converge towards lowest energy states till an optimum point (Ojugo et al, 2013). The algorithm is as thus:

Simulated Annealing Algorithm {}

Initialize individual state, energy and temperature

Loop until temperature is at minimum

Loop until maximum number of iterations reached

Find neighbourhood state via neighbourhood function

If neighbourhood state has lower energy than current

Then change current state to neighbouring state

Else if the acceptance probability is fulfilled

Then move to the neighbouring state

Else retain the current state

Keep track of state with lowest energy

End inner loop: End outer loop

ANN first yields candidates of low fitness in training, and if better solutions are not found, best individuals are chosen after a number of runs, until optima is found. ANN uses its exploratory search of multiple individuals; while SA uses it flexibility in finding a global optima (Ojugo et al, 2013c).

Factors to be defined: (1) ANN: number of runs, dataset for calibration, population representation of dataset, size and cross validation function; And (2) SA (with ANN complete), what neighborhood size and energy function is employed to choose fit candidates till solution is found. Temperature schedule is applied that randomly re-initialize network for series of Markov chain. Neighbourhood function is applied to randomly changed individual energy. The fitness function is recomputed to track individuals of low energy state but with threshold value of 0.8 to enter SA early enough to apply temperature schedule needed. Thus, a moderated Markov chain is used that accepts the states with energies of lower or equal to current state's energy. It runs till state of energy 0 is reached (solution is found). SA and ANN, shares the same fitness function (Ojugo et al, 2013c).

3.3 Hybrid Gravitation Search Neural Network (GSANN)

GSA is based on Newton's laws of gravity and motion with its main idea, being to consider isolated system of masses, where every mass represents a solution to a certain problem. Law states that particles attract each other with gravitational force acting between particles that are directly proportional to product of their masses and inversely proportional to the distance between them (Rashedi et al, 2009a). Thus, agent's performance depends on its mass as they attract each other via gravitational pull towards those of heavier mass. Agents are randomly initialized with gravitational force defined as:

$$G(t) = \frac{M_i(t) * M_j(t)}{Rij(t) + \varepsilon}\{Xj(t) - Xi(t)\} \quad (1)$$

R_{ij} is Euclidean distance between masses for i and j, G(t) is gravitation force at t with small constant ε – which decreases at t, to control pool and search's accuracy. Total force is:

$$F_i^d = \sum_{j \in kbest, j \neq 1} rand(i) * Fij \quad (2)$$

Acceleration at t, in d dimension is directly proportional to force on agent i, and inversely proportional to agent's mass:

$$A_i^d(t) = \frac{F_i^d(t)}{Mij(t)} \quad (3)$$

Next agent velocity is a function of its current velocity plus current acceleration – to update next position given by X as:

$$V_i^d(t+1) = rand(i) * V_i^d(t) + A_i^d(t) \quad (4)$$

$$X_i^d(t+1) = X_i^d * V_i^d(t+1) \quad (5)$$

Mass is updated as fitness value of agent i at time t given as:

$$Mi(t) = \frac{Fit(i) - worst(t)}{best(t) - worst(t)} \quad (6)$$

Strongest agents from their fitness route for maximization task is given as:

$$worst(t) = \underset{j \in \{1,2 \ldots N\}}{\text{Maximize}} Fit(t) \quad (7)$$

$$best(t) = \underset{j \in \{1,2 \ldots N\}}{\text{Minimize}} Fit(t) \quad (8)$$

At start, agents are located as solution trained in ANN, and passed over to GSA. With each cycle, velocity/position is updated via Eqs. 4/5; while G/M is found via Eqs. 1/6. The model stops if an optima is found or stops using its stop criterion (computational expensive). GSA uses exploration ability to navigate and guarantee its choice value for random agents, and exploitation ability to allow agents of heavier masses move slower in order to attract those of lesser mass as well as locate optima, around a good solution in the shortest time possible (Rashedi et al, 2009b and Ojugo, 2012a). Its algorithm is as thus:

Input: Audit data, generations and population size.

Randomly initialize created agent (rule) for search space identification

Compute fitness function of agents and created rules as thus

Set w1 = 0.2, w2 = 0.8, MaxGenerations = 60 (epoch)

Set n =200, generationCounter = 0, created rule groups = 30

For ANNGSA Training, each rule set:

Set A = 0, AB = 0

AB = AB + 1 //AB++

If record matches only condition part

A = A+1 //A++

End if

Select any 30-fitted rules, 1 for each group to form new pool

Apply G(t), best(t), worst(t) and Mi(t) for i=0

Update G(t), best(t), worst(t) and Mi(t) for i=1,2,3,…n:

Randomly select 3-fitted rules based best fitness value

Compute total force in all directions

Compute acceleration and velocity of rules in space

Update each rule's M, G and Position

Select best chromosome to be parent

Repeat Process 14 until stop criterion is reached

End

ANN first yields candidates of low fitness in training, and if better solutions are not found, best individuals are chosen after a number of runs, until optima is found. ANN uses its exploratory search of multiple individuals; while GSA uses it flexibility in finding a better optimal point, even when local maximas are present (Ojugo et al, 2013c). Factors that must be defined: (1) ANN: number of runs, dataset for calibration, pool representation of dataset, size and cross validation function; candidates. The fitness function is recomputed to track individuals of value 0.8 – though, ANN finds individuals of low fitness with G, M and position of agents (rules) updated to enter GSA early enough till state of energy 0 is reached – implying a solution is found (Ojugo et al, 2013c).

3.4 Experimental Model Validation

Figures 4-7 describes the accuracy with which each hybrid model captures genuine malware attacks and its corresponding ***false***-positive (error rates in classifying of malware signatures as emergent in each intrusive attack) and ***true***-negatives (error rates in not accurately classifying malware signature emergent in the intrusive attack – which can arise from code signature variation in the virus and due to code obfuscation method used by the metamorphic engine).

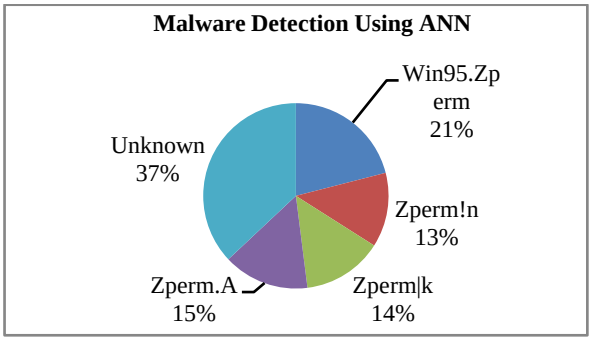

Figure 4: Evolved Variants Malware IDS Using ANN

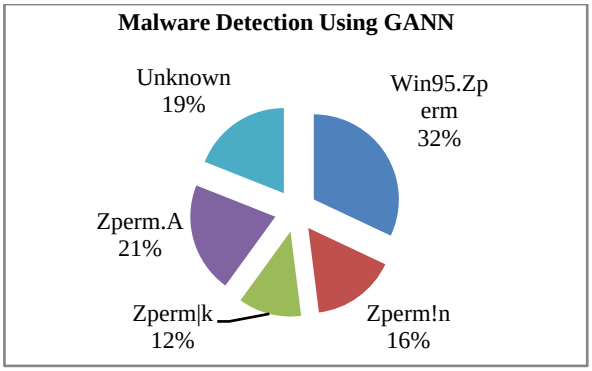

Figure 5: Evolved Variants Malware IDS Using GANN

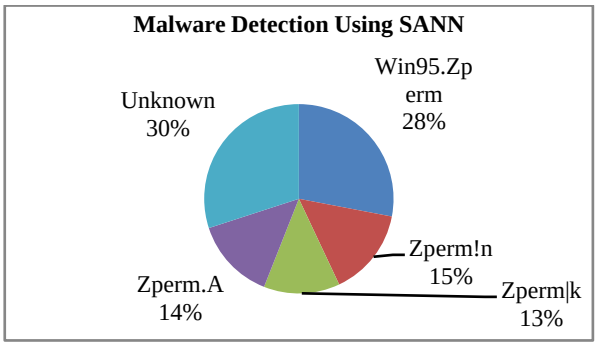

Figure 6: Evolved Variants Malware IDS Using SANN

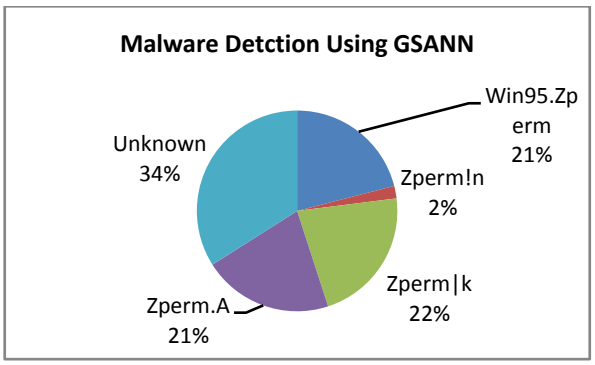

Figure 7: Evolved Variants Malware IDS Using GSANN

4 RESULT FINDINGS AND DISCUSSION

Using a simple Python implementation of various matching algorithms, algorithms were compared based these feats:

4.1 Classification Accuracy

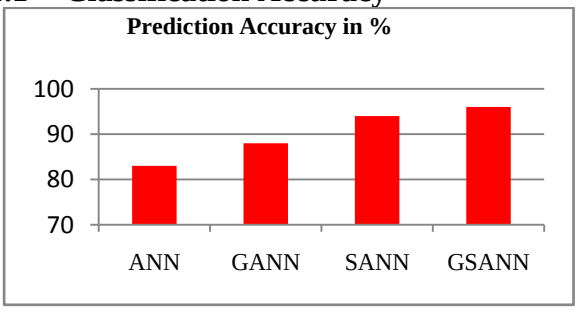

Figure 8: Prediction Accuracy of Algorithms in percentage

4.2 Processing Speed

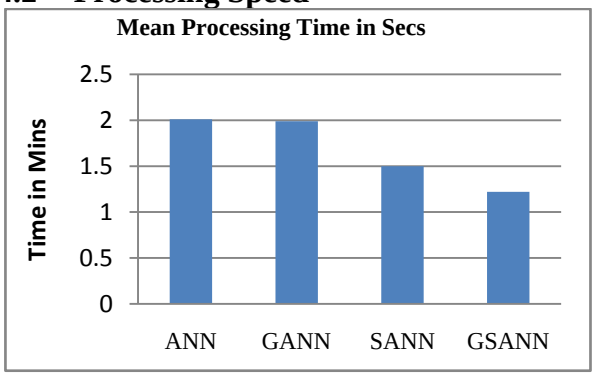

Figure 9: Processing time in Seconds

4.3 Convergence Time

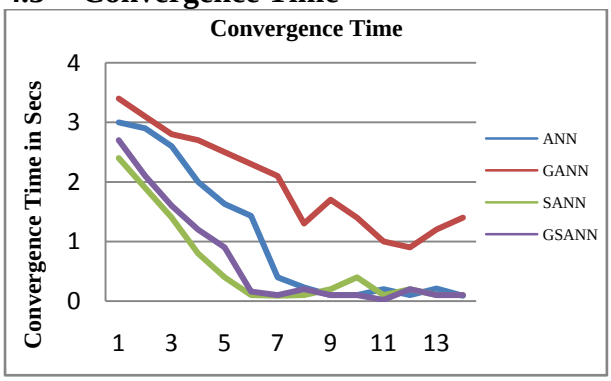

Figure 10: Convergence time of matches

Figure 10 shows the convergence time of matches in which the various algorithms took in finding the patterns within the text.

4.4 Discussion of Findings

Top rules have same fitness range [0.8, 0.8065] and are estimated 80% good to be used in detection – to imply the achievement of generating a bunch of good rules, rather than a single optimum rule – is better in intrusion detection. 10-out-of-22 rules have destination port as -1, so that the rules looks out for connections from any destination port. This increases the chances of detecting intrusion, improves the generality of rules, and provides for new attack types and its corresponding rules to be added to knowledgebase. The rule generator used a population of 400, w1 = 0.2, w2 = 0.8, 5000 epoch-evolutions and 0.05 probability of a gene to be mutated respectively.

After training and testing models, the results are thus:

4.4.1 *ANN* was run 15 times (to eradicate biasness) and took 102seconds to find optima after 329-iterations (at best). It was able to generate each time, multiple local maxima (good rules) and its time varied between 102seconds and 70minutes. Convergence time depends on how close initial population is to solution and on mutation applied to individuals in the pool.

4.4.2 *GANN* was run 15 times (to eradicate biasness) and took 89seconds to find optima after 280 iterations (at best). It was able to generate each time, multiple local maxima (good rules) and its time varied between 102seconds and 70minutes. Convergence time depends on how close initial population is to solution and on mutation applied to individuals in the pool.

4.4.3 *SANN* was run 15 times, took 62seconds to reach optima after 380 iterations. It generated at intermittently multiple local maxima (good rules) and its time varied between 62 seconds and 40minutes. Convergence time depends on initial population, the temperature schedule applied and series of random walks applied to pool. It is to be noted that SA is most useful in the generation of best rule (and not set of better rules, goal of this study).

4.4.4 *GSANN* was run 15 times (to eradicate biasness) and took 102econds to find optima after 322 iterations (at best). It was able to generate each time, multiple local maxima (good rules) and its time varied between 89 seconds and 54minutes. Convergence time depends on fitness function of weights with G, M and position updates of rules and individuals in the pool

4.5 Rationale for Choice of Algorithms

Most mathematical, machine learning models are inspired by evolution, biological and behavioural population. They search a space via hill-climbing method which is flexible, easily adapts to changing states and suited for real-time app to guarantee high global convergence in multimodal task. Initialized with random pool, it allocates increasing trials to regions of high fitness to find optima. Once a peak is found, model restarts with another randomly chosen, starting point. Its simplicity, well suited for dynamic feats/phenomena of many local maxima – makes them appropriate. Each random trial is done in isolation and as search progresses, it allocates its trials evenly over space and still evaluates as many points in regions found to be of low fitness as in regions found to be of high fitness. Its demerit is its inadequacy for linear model with small regions surrounded by low fitness – making such functions/models, difficult to optimize.

4.6 Implementation Tradeoffs

Result trade-offs are as follows (Ojugo et al, 2013):

4.6.1 *Result Presentation:* Researchers often display flawed and unfounded results, to validate new/modified model rather than re-test limitations, insufficiency, biasness and inabilities of existing ones. This is because negative results are less valuable and most of such models aim to curb the non-linearity and dynamism in the phenomena they are predicting alongside discovering feats and underlying properties of the historic datasets used, to train, cross validate and test such models.

4.6.2 *Efficiency:* modeler sand researchers can often use figure to show how well their prediction is quite in agreement with observed values (even with their limited dataset used for training the model that is often times squeezed). Some plot for observed and predicted values are often not easily distinguishable – as such modelers do not even provide numerical data to support their claim for their system (though their model is in good agreement with observed values). Some measure of goodness does not provide the relevant data.

4.6.3 *Insufficient Testing:* Validation compares observed on predicted values. Many studies suffer

from inadequate dataset. If model aims to predict dynamic state, such ability should not be demonstrated with misleading results of limited dataset; and inconclusive and unclear contributions. Model must be adequately tested with methods laid bare so that process can be repeated to validate the usefulness and authenticity of such models.

4.6.4 *Model validation:* is not an undertaking to be carried out by a researcher or research group; but rather, a scientific dialogue. Improper model applications and ambiguous results often impede such dialogue. This study aims to greatly minimize confusion in study of model as well as their corresponding implementation in IDS.

5 CONCLUSION AND RECOMMENDATION

Models serve as educational, predictive tools to compile all existing data about a task, serve as a vehicle to communicate hypotheses, a means to investigate parameters crucial in estimation and help us better understand a problem domain. Simple models may not provide enough new data, whereas very complex models may not be fully understood. A model's use and application as an intellectual tool requires less accurate numerical agreement in prediction. But rather requires feedback mechanisms, as more important – as only models that are understandable and manageable, can be fully explored. A balance of complexity and simplicity is crucial to studying the relevant processes of how the model works.

Study implements ruled-based IDS used to generate a set of classification rules from DARPA audit data used for training. The support-confidence fitness function is used to evaluate goodness of each rule. Based on weight selection (fitness values w1/w2), generated rules can detect network intrusions, detect anomalies and classify attack types.

The research frameworks are divided: training and testing. Study implements IDS with Java and C# on Linux platform (due to program interaction, simplicity, code execution, speed and connection ease). All critical factors in IDS, as a system must go through millions of network connection – inspecting each to determine intrusive and non-intrusive connections.

Study limitations include: (a) generated rules were biased to training data. This was resolved by carefully selecting the number of generations at training to avoid overtraining or the number of top best-fit rules so as not to over fit model, and (b) support-confidence framework may be simple to implement and provide improved accuracy to final rules, but it requires the whole training data to be loaded into memory before any computation. For very large training datasets, it is neither efficient nor feasible, and system requires enough cache memory to hold the data to be processed.

6 REFERENCES

[1] Aarts, E.H., Korst, J and Van Laarhoven., (1997). *Simulated annealing"* as in Aarts, E.H and Lenstra, J.K., (eds.) *Local Search in combinatorial optimization*, John Wiley and Sons.

[2] Abramson, D., Dang, H and Krishnamoorthy, M., (1996). *Simulated annealing cooling schedules for school timetabling problem*, Asian Operation Research, 3(5), p11-24.

[3] Alpaydin, E., (2010). *Introduction to Machine Learning*, McGraw Hill publications, ISBN: 0070428077, New Jersey

[4] Al-Anni, M. K. and Sundararajan, V., (2009): *Detecting a denial of service via AI tools and GSA*, Indian J. of Science, 2(2), p16-21.

[5] Axelsson, S., (2004). *Combining a Bayesian Classifier with Visualization*: *Understanding IDS*, VizSEC/DMSEC'04, ACM-1581139748/04/0010.

[6] Bacchus, F., (2010). *Constraint satisfaction problem*", Computer Lecture notes, cs.toronto.edu/~ fbacchus/. Last access Feb. 13, 2013.

[7] Bashir, H.A and Neville, R.S., (2013). *Hybrid evolutionary computation for continuous optimization*, arxiv: 1303.3469, http://arxiv.org>cs

[8] Bayram, H and Sahin, R., (2013). *A new simulated annealing approach for the traveling salesman problem*, Mathematical and Computational Applications, 18(3), pp 313 – 322.

[9] Brailsford, S., Potts, C.N and Smith, B.M, (1998). *Constraint satisfaction problem: algorithms and applications*, European J. of Operation Research, 119, p557-581.

[10] Chittur, A., (2001): *Model generation for an intrusion detection system via GA*, hacktory.cs.columbia.edu/sites/default/files/gaids-thesis01.pdf.

[11] Chou, T.S., Yen K.K and Lou, J., (2008): *Network intrusion detection design using feature selection of soft computing paradigms*, World Academy of Science, Engineering and Technology 47.

[12] Coddington, P., (2012). *Constraint satisfaction problems*, Computer Lecture notes, cs.adelaide.edu.au Last accessed Feb 13, 2013.

[13] Crosbie, M., and Spafford, G., (1995): *Applying genetic programming to intrusion detection*,www.aaai.org/Papers/Symposia/Fall/1995/FS-95-01/FS95-01-001.pdf.

[14] Darrall, H., Jacobson, S.H and Johnson, A.W., (2003). *Theory and practice of simulated annealing*, Handbook of Metaheuristics, Springer, ISBN: 978-1-4020-7263-5, p287-319.

[15] Diaz-Gomez, P. and Hougen, D., (2005): *Improved off-line intrusion detection using a GA*, cameron.edu/~pdiaz-go/Art_ICEIS.pdf

[16] Dos Passos, W., (2013). *Numerical methods, algorithms and tools in C#*, Taylor and Francis Inc., ISBN: 9780849374791.

[17] Fausett, L., (1994):*Fundamentals of Neural Networks*, Prentice Hall: USA, ISBN: 0133341860.

[18] Gong, R.,Zulkernine, M. and Abolmaesumi, P., (2005):*A software implementation of GA based approach to network intrusion detection*, www.cse.msu.edu/~cse848/Studentpapers/Tavon_Pourboghrat.pdf

[19] Harrington, P., (2012). *Machine Learning in action*, Manning publications, ISBN: 9781617290183, New York

[20] Johnson, D., Aragon, C, McGeoch, L and Schevon, C., (1991). *Optimization by simulated annealing: an experimental evaluation graph partitioning*, Operation Research, 39(3), p865-892

[21] Kandeeban, S. S. and Rajesh, R. S., (2007): *GA for framing rules for intrusion detection*, J. Comp. Sci and Security, 7(11), p285-290.

[22] Kandeeban, S. S. and Rajesh, R. S., (2010): *Integrated intrusion detection system via soft computing*, J. Network Security, 10(2), p87

[23] Kayacik, H., Zincir-Heywood, A and Heywood M., (2005): *Selecting features for IDS: a feature relevance analysis on KDD 99 dataset*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.7574&rep=rep1&type=pdf

[24] Kirkpatrick, S., (1983). *Optimization by simulated annealing*, Science, 220, p671-680.

[25] Kitanidis, P and Bras, R., (1980). *Real-time forecasting with a conceptual hydrologic model, applications and results*, Water Resources, 16(6), pp.1034–1044.

[26] Kurose, J. F. and Ross, K. N., (2010): *Computer network a top down approach*, Pearson publisher, ISBN-10: 0-13-136548-7.

[27] Lassig, J and Sudholt, D., (2011). *Adaptive population model for offspring population and parallel evolutionary algorithms*, arxiv: 1102.0588

[28] Lavender, B., (2010): *Implementation of GA into IDS and integration into nprobe*, http://brie.com/brian/netga/Lavender_Report.pdf.

[29] Li, W, (2004).*GA approach to network IDS*,security.cse.msstate.edu/docs/Publication/wli/DOECSG2004.pdf

[30] Mitchell, T.M., (1997). *Machine Learning*, McGraw Hill publications, ISBN: 0070428077, New Jersey.

[31] Newman, M.E, (2003). *The structure and function of complex networks*. SIAM Reviews, 45(2), p167.

[32] Nikolaev, A.G., Jacobson, S.H., Hall, S.N and Henderson, D., (2011). *Framework for analyzing suboptimal performance of local search algorithms*, Computation Optimization and Applications, 49(3), p407.

[33] Ojugo, A.A., (2005). *Comparative SA model to solving optimization problem – case of virus propagation on time varying networks"*, Unpublished MSc, Nnamdi Azikiwe University Awka, Nigeria.

[34] Ojugo, A.A., Eboka, A.O and Yoro, R.E., (2007). *Hybrid simulated annealing neural network to solving Sudoku*, Proceedings of 4th IRDI Conf. on Science Tech, p78, Uyo: Nigeria.

[35] Ojugo, A.A., (2012a). *Hybrid artificial neural network gravitational search algorithm for rainfall runoff*, Unpublished PhD Thesis, Dept. Computer Science, Ebonyi State University Abakiliki, Nigeria.

[36] Ojugo, A., Eboka, A., Okonta, E., Yoro, R and Aghware, F., (2012b). *GA rule-based intrusion detection system*, J. of Computing and Information Systems, 3(8), p1182.

[37] Ojugo, A.A., (2013a). *Virus propagation on time varying graphs*, Technical-Report, Centre for High Performance and Dynamic Computing (CHPDYC), TRON-03-2013-01, Federal University of Petroleum Resources, Nigeria, p24-37.

[38] Ojugo, A.A., and Yoro, R., (2013b). *Computational intelligence in stochastic solution for Toroidal Queen task*, Progress in Intelligence Computing Applications, 2(1), doi: 10.4156/pica.vol2.issue1.4, p46

[39] Ojugo, A.A., Emudianughe, J., Yoro, R.E., Okonta, E.O and Eboka, A.O., (2013c). *Hybrid artificial neural network gravitational search algorithm for rainfall runoff*, Progress in Intelligence Computing and Applications, 2(1), doi: 10.4156/pica.vol2.issue1.2, p22.

[40] Olusegun, F., Oluwatobi, O. A. and Adewale O. O., (2010): *ID-SOMGA: self organising migrating GA-based solution for Intrusion Detection*, Computer and Information Science, 3(4), p80

[41] Perez, M and Marwala, T., (2011). *Stochastic optimization for solving Sudoku*, Proceeding of IEEE on Evolutionary Computing, p256 – 279.

[42] Perez, M and Marwala, T., (2012). *Microarray data feature selection using hybrid genetic algorithm simulated annealing*, IEEE conference on Electrical and Electronics Engineers, doi: 10.1109/EEEI.2010.6377146, pp 1 – 5

[43] SalehElmohamed, M.A, Fox. G and Coddington, P., (1998). "*A comparison of annealing techniques for academic course scheduling*", Notes on Intelligence Computing, DHCP-045, p1-20. www.dhpc.adelaide.edu.au.

[44] Schafer, J.D., (1985): *Multiple objective optimization with vector evaluated Genetic Algorithm*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.5689&rep=rep1&type=pdf.

[45] Shanmugam, B. and Idris, N. B., (2011): *Hybrid intrusion detection systems using fuzzy logic*, www.intechopen.com/download/pdf/14361.

[46] Sontag, E.D., (1998). "*Learning for continuous-time recurrent neural networks*", Systems and Control Letters, 34, pp. 151-158.

[47] Sorkin, G., (1991). *Theory and practices of SA on special landscape*, PhD thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley.

[48] Thomson, J and Dowsland, K., (1995). *General cooling schedules for SA based timetable problems*, Proceeding of Practice and Theory of automated timetabling, Edinburg: Napier University, pp421–444 Vollmer, T., Alves-Foss, J. and Manic, M., (2011). *Autonomous rule creation for intrusion detection*, inl.gov/technicalpublications/Documents/5025964.pdf