Vol 8. No. 1 Issue 2 – May, 2015

African Journal of Computing & ICT

www.ajocict.net

An Intelligent Hunting Profile for Evolvable Metamorphic Malware

A.A. Ojugo
Dept. of Mathematics/Computer
Federal University of Petroleum Resources
Effurun, Nigeria
ojugo.arnold@fupre.edu.ng

A.O. Eboka
Department of Computer Education
Federal College of Education (Technical)
Asaba, Delta State, Nigeria
andre_y2k@yahoo.com

ABSTRACT

Malware are destructive by altering host machine's behaviour as it self-replicates its codes unto the host's files. Some, have the ability to change its structure on execution via mutation and other code obfuscation – to generate complex variants with same functionality; But, different in their syntax and signature. This renders signature-based detection quite unreliable and their detection, tedious. Our study generates evolved complex variants of the Zmist malware using memetic algorithm, we then create an effective profile and rules trained via the Hidden Markov model, that efficiently detect the Zmist variants with high (classification) probability; And thus, drastically reduce false-positive and true-negative rates. We use HMM clustering ability to explore sample cluster-profiles of the Zmist metamorphic engine to help us learn the underlying code clusters from sample data, and navigate its engine to yield faster and completely morphed variants of Zmist using memetic algorithm. Evolved variants of were tested on and against commercial antivirus.

Keywords— Intelligent, stochastic, hunting, profile, evolutionary, clustering, markov model, virus, malware

African Journal of Computing & ICT Reference Format:
A.A. Ojugo & A.O. Eboka (2015). An Intelligent Hunting Profile for Evolvable Metamorphic Malware.
Afr J. of Comp & ICTs. Vol 7, No. 3. Pp 181-190,

1. INTRODUCTION

Computer 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 [1][2][3].

[4] in [5] notes the first computer virus was a boot sector virus created in 1986 that infects the host machine resources such as files and macros, operating system, system sectors, companion files and source code. Use of Internet for data transfer has become a soft target for their widespread to wreak havoc faster globally. Early detection of viruses is thus, imperative to minimize the damage caused.

A. Modules of a Computer Virus

[5] Virus has 3-modules: infect, trigger and payload. Infect is the mechanism to modify its host and contain copies of it. Trigger details when and how to deliver payload (details the damage to be done). Trigger and payload are optional. Fig. 1a is virus pseudo-code; while Fig. 1b is an infect pseudo-code. [6] Subroutine *Infect* selects a target from M-targets to infect when run. *Select_target* details target selection criteria as same target should not be repeatedly selected; else, reveals presence of a virus. And, *Infect_code* performs actual infection by inserting its code into the target.

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

Fig 1a: Virus Pseudo-code

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

Fig 1b: Infect Pseudo-code

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 [7].

Viruses are classified as:

a. ***Simple*** virus replicates itself on execution in a host machine so that it gains control, attaching a copy of itself to host machine's files or program as it spreads. It then transfers back control to host program afterwards. It is easily detected by scanning for a defined sequence of bytes called signature [8].
b. ***Encrypted*** virus scrambles its signature to make it unrecognizable. 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 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 the virus appears different, from one infection to another. This makes signature detection technique difficult so that the antivirus must scan for a constant decryption routine instead [9].
c. ***Polymorphics*** consists of scrambled body, its mutation engine and decryption routine. Its decryption routine first acquires control of host machine as it decrypts scrambled body and mutation engine. It then transfers control to the now unscrambled body to locate new file to infect – unto which it copies its body, also copying unto the RAM its mutation engine and invoking its mutation engine to randomly generate a new decryption routine to decrypt its body with little or no semblance to the previous decryption routine. It then appends this newly encrypted body, a mutation engine and decryption routine to the newly infected file – so that the encrypted body and decryption routine, varies from one infection to another. It has no signature and decryption routine – making any two infections ***not*** alike [10].
d. ***Metamorphics*** avoid detection by completely rewriting its code each time it infects new file. It accomplishes code obfuscation and meta-morphing, which is 90% of its assembly codes [8-10].

B. Virus Detection Mechanisms

Antivirus software detects, prevent and remove all malware, including but not limited to viruses, worms, Trojans, spyware and adware. Antivirus use strategies namely: heuristic search, cyclic redundancy check, logic search and spy on processes to scan for viruses. Detection mechanism is broadly grouped into: (a) signature-based scans for signature, and to evade it – virus makers create new virus strings that can alter their structure while keeping its functionality via code obfuscation method, and (b) code emulation creates sandbox or virtual machine, so that files are executed within it and scanned for virus. Once the virus is detected, it is no longer a threat – since it is running in controlled environment that limit damage to host machine [5, 11-12].

Antivirus often impairs system performance, and incorrect decision may lead to security breach as it runs at the kernel of the operating system. If an antivirus uses heuristics, its success depends on the right balance between positives and negatives. Today, malware may no longer be executables. Macros can present security risk and antivirus heavily relies on signature-detection. Metamorphic and polymorphic viruses, evades and makes signature detection, quite ineffective [13]. Studies have shown that anti-virus effectiveness decreased against unknown or zero-day attacks. This problem has been magnified by the changing intent of virus makers. Independent testing on all the major virus scanners consistently shows none to yield 100% detection. The best ones yield 99.6% detection, while lowest is 81.8%. Thus, all scanners can yield a false positive result as well so that they identify benign files as malware [14-15].

C. Metamorphic Malware or Viruses

Rather than use encryption, metamorphics change its code structure/appearance while keeping its functionality. It does this via code obfuscation methods as in fig 2. 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 [4-5, 9, 16-17].

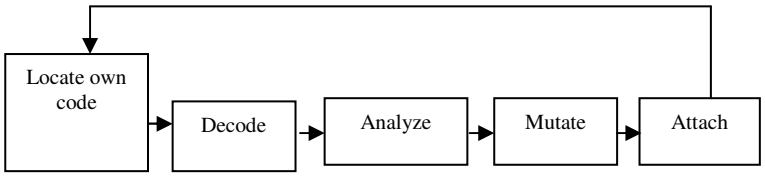

Fig. 2: Distinct Signature of Metamorphic Virus

[18-19] 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) ***expander*** replaces an instruction with code set to perform same action, (d) ***swapper*** reorders these codes by swapping two/more unrelated codes, (e) a ***relocator*** assigns and relocate relative references like jump/call, (f) ***garbager*** (constructor) inserts whitespaces (do-nothing codes) and (g) the ***cleaner*** (destructor) undoes the actions of a garbager by removing whitespaces instructions. [9, 17, 20] 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.

D. 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 [21]. Code obfuscation is achieved via [22-23]:

a. Register Usage Exchange/Rename modifies 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 whitespaces, which do not affect its code 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 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 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.

2. MATERIALS / METHODOLOGY

A. *Virus Abstract Representation*

The study uses the Zmist metamorphic engine (whose rules or heuristics uses substitution, transposition and trash – all of which are permutation) methods to build viruses of the same functionality. The engine changes its opcode, generating new variants from old versions (authored by Zombie and extracted from [24]. Zmist at its release, was one of the most complex binary viruses ever written, which uses Entry-Point Obscuring that supports a unique code integration scheme, and occasionally inserts jumps after each instruction in a section, pointing to the next instruction. It extremely modifies files from one generation to next via camouflage, which makes it suited for the study [7, 19].

B. Machine Learning / Evolutionary Models

Statistical pattern analysis has proven a successful technique to detect metamorphics via machine learning (soft-computing) paradigms. Soft computing is an inexact science that uses evolutionary optimization models to resolve tasks. It achieves its tractability via optimization by exploiting historic data as well as exploring human knowledge encoded via statistical pattern analysis, mathematical models and symbolic reasoning [25] to perform quantitative data processing and yield qualitative knowledge as its new language. The models are tuned to be robust, so that even with partial truth, imprecision, uncertainty and noise applied to its input, it yields an output guaranteed of high quality. They are mostly inspired by behavioural patterns and evolution in biological population as well as natural laws. They explore 3-basic feats as they seek to unveil the underlying probability of data feats of interest namely: (a) ***adaptation*** yield agents void of local minima and with high-diverse random migrantion introduced into the model to slow its convergence and create a balance between exploitation and exploration so that learning feats of change, biases its solution accordingly, (b) ***robustness*** estimates its effectiveness of the model as employed in the task at hand, and (c) decision is ***flexible*** as uncertainty feats impacts on the model's future state continually in forecasts while focusing on its goal state and its ease integration [26]. Example include models such as genetic algorithm, firefly algorithm, neural networks, Markov model etc – all known tools and recently, used in hunting cum effective detection of polymorphics cum metamorphics.

C. Hidden Markov Model

[27] Consider a series of state and its associated probabilities to each transition between states. Such state (chain) is a Markov, if the transition probabilities depend only on the current state (not on previous states) such that the Markov chain has no memory. More precisely, it is a first order Markov chain if the nth order Markov chain depends both on current and its $n–1$ previous states. Also, the Markov chain has no finite memory. The Hidden Markov model has been successfully employed in the studies of bioinformatics as well as molecular biology for gene sequencing.

Thus, we represent a simple DNA sequence using a Markov chain process as: P_{AT} is probability of the transition from state A to T; while P_{TA} is transition probability from state T to A, and so on respectively given that in the DNA chemical code, A = Adenosine, C = Cytosine, G = Guanine, and T = Thymine as in Fig 1.

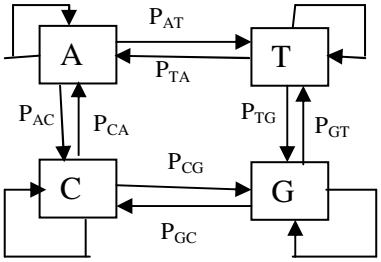

Fig 3: State Transition / Automata

Each arrow represents transition probability of a specific base followed by another base. Transition probability is calculated after observing several DNA sequence and corresponding transition probability matrix yields a compact representation of the transition probabilities – noting that a Markov chain process leads to a corresponding Markov model; And, that each event depends only on the previous event. Transition probability from state of observed symbol *s* to another state *t* is given by Eq. 1 [28]:

$$a_{st} = P(x_i = t \mid x_{i-1} = s) \text{ for } 1 \leq s, t \leq N \quad (1)$$

where
N is the number of states and x_i is the state at step *i*).

The sum of transition probabilities from each state equals 1 since these transitions represent a probability distribution as the probability associated with each step of/in the model. Following Bayes Theorem, the probability of the sequence relative to the given model is computed using Eq. 2 noting that $P(x_i)$ is probability of starting at first state x_i, and 'begin'/'end' state helps accommodate first/last symbols of output sequence.

$$\begin{aligned} P(x) &= P(x_L, x_{L-1}, x_{L-2} \ldots, x_1) \\ &= P(x_L \mid x_{L-1}, x_{L-2} \ldots, x_1)\, P(x_{L-1}, \mid x_{L-2} \ldots, x_1) \\ &= P(x_L \mid x_{L-1})\, P(x_{L-1} \mid x_{L-2}) \ldots P(x_2 \mid x_1)\, P(x_1) \\ &= P(x_i) \prod_{i=2}^{L} a x_{i-1} x_i \quad (2) \end{aligned}$$

[28] Also, current event depends on more than one previous event. An *n*th order Markov chain on *m* symbols is represented as a first order Markov chain with *mn* symbols. Thus, given a series of observations (i.e. output sequence) from a Markov process, we wish to determine which state generated each observation. Consider N buckets with a given distribution of coloured balls in each. Note that we are well aware of the distribution of the balls in each bucket as well as the rule for determining which of the bucket to select from.

Being a Markov process, this rule for choosing the bucket from which we can select from depends on the previous selection. [29] Suppose, a sequence of colours correspond to balls selected; But, we do not know which buckets they were selected from. That is, the Markov process itself is hidden – we would like to gain information about this hidden process through the observations – that is, the colour of the balls selected. So far, we only outlined the basic structure of a hidden Markov model, the problem can be solved via hidden Markov model approach using this simple example where:

- ✓ O: the observation sequence
- ✓ T: is the length of the observation sequence
- ✓ N: number of states in the HMM process
- ✓ α – is the alphabet for the model
- ✓ M: number of symbols in the alphabet
- ✓ A: the state transition probability matrix
- ✓ a_{ij} probability of the state transition from *i* to *j*
- ✓ $b_i(k)$: probability of observing *k* in the state *i*
- ✓ B: probability distribution of the observations (one distribution for each Markov process)
- ✓ λ = (A,B,π) and it represents the HMM

HMM is given by λ = (A,B,π) where the matrices of A,B and π may or may not be known, depending on the task. Thus, the model can be suited for any of the following tasks [29]:

a. **Problem 1**: Given observations and parameters N and M, determine model λ = (A,B,π) that best fits sequence. We train model to fit data. HMM training requires no *aprior* assumptions about the model other than outline parameter N and M, which specifies the size of the model.
b. **Problem 2**: Compute probability that the given model produces an observation sequence if given the model λ – (A,B,π) and an observation sequence O, compute P(O/ λ).
c. **Problem 3**: Uncover HMM λ = (A,B,π) and observation sequence O to determine most likely sequence of states X = (x_1, x_2, …, x_T) that could have produced the sequence.

Suppose – in the example of gene sequencing, we have CTAG states, which can occur in any number of sequence and in any number of times *T*, to generate an observation between the states. We have that number of symbols M = 4 (i.e. CTAG). If our observed sequence O = {C,T,G,A,T,T,G,G,A} from the base sequence (which is made visible); However, we seek to uncover the signature of the new variants sequence when we program this sequence to yield a new observation. But, we require these 4-symbols too to yield the new variants (unsure sequence) as many as 3-times the sequence we see clearly. Thus, there are also 4-hidden states so that N = 4 (for normal and hidden sequence).

Our transition probability matrix is computed and represented by fig 4. From this, we have that the ratio of the base virus to the completely morphed copy yet to be generated as it infects new system is given by 1:3 and using the Maximum Expected Likelihood model, we arrive at this transition probability matrix as in fig 4. Also, we have the initial distribution π which specifies the probability that the Markov process begins with the normal observed sequence as well as a biased generated variant (after infecting a body or system respectively) – so that π = [0.3 0.7]. The values for (A,B,π) are all row-statistic – that is, each row is a probability distribution. Note that the series of states in the underlying Markov process is hidden and we observe that the sequence of CTAG that result from the process are assumed as N and M, which in this case are known [28].

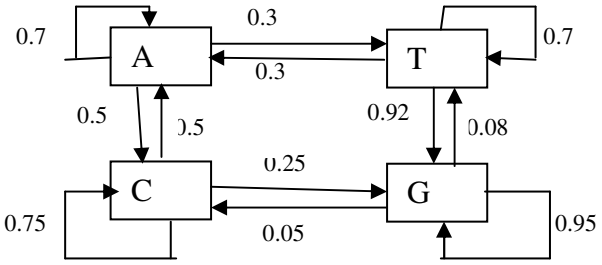

Fig 4: State Transition or Automata Diagram

D. The Bayesian Profile Hidden Markov Model

The Hidden Markov Model is a double embedded chain that models complex stochastic processes. The Markov process is a chain of states with probabilities associated to each transition between states. In *n*-order Markov, its transition probabilities depend on ***current*** and ***n-1 previous*** states. A Hidden Markov model process determines the state generated for each state observation in a series (solution space or output sequence). In malware, an instruction not accepted by the trained HMM, yields high probability of being a ***malware*** [29]. Traditional HMM scores data through clustering based on the profile values. The probabilities of initial set of instructions are sampled – then checked to see if such instructions are metamorphic viruses. HMM maintains log in memory to help reduce true-negatives (instructions that behave malware-like) and false positives (unclassified variants of malware). Thus, our HMM is initially trained to assimilate normal behaviour of Zmist metamorphic engine. It then creates a profile of the malware codes, which it classifies into low, medium and high profile range [28-29, 41].

However, profile HMM as a variant of HMM, aims to deal with the fundamental problems of the HMM by: (a) it makes explicit use of positional (alignment) data contained in the observations or sequences, and (b) it allows null transitions, where necessary so that the model can match sequences that includes insertion and deletions [27-29]. In malware detection, O is each code of metamorphic engine denoted as a rule, T is time each code takes to execute, N is number of codes in sequence and obfuscation methods used as etched into HMM, M is number of code access to registers contained in Zmist engine, π is initial state (starting code) for Zmist engine, A is the state transition probability matrix, a_{ij} is probability of a transition from state *i* to another state *j*, B contains *N*-probability distribution codes in knowledgebase from where profiles are been created (one code for each state of the process); while HMM λ = (A,B,π). Though, parameters for HMM details are incomplete as above; But, the general idea is still intact.

www.ajocict.net

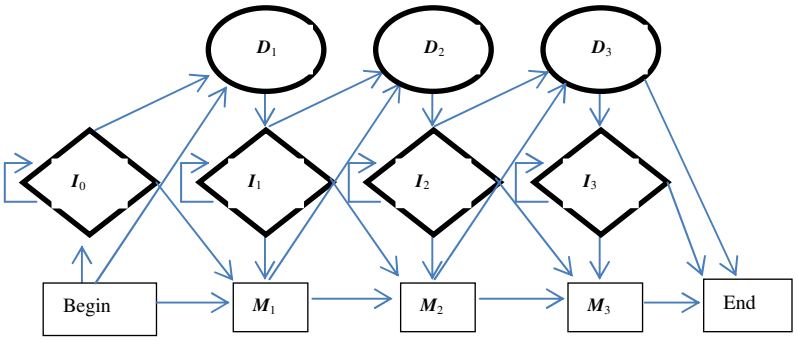

Fig 4: PHMM with 3-Match States

We can also align multiple codes (data) rules as sequence with significant relations. Its output sequence determines if an unknown code is related to sequence belonging to either of the Zmist variant class (or not) as comprise in the Bayesian net. We then use the profile HMM to score codes and make decision. The circles are ***delete*** state that detects classified Zmist codes in the knowledgebase, ***diamonds*** are insert states that allow us ***sandbox*** codes that are unclassified upon which the knowledgebase is updated for classified false-positives and true-negative detection; while ***rectangles*** are matched states that accurately classifies codes into Zmist variants as in the standard HMM [28-29].

Match and insert are emission states in which an observation is made as PHMM passes through all the states. Emission probabilities, corresponding to B in standard HMM model is computed based on frequency of symbols (in this case, Zmist codes) that can be emitted at a particular state in the model; But, are positional-dependent (in contrast to standard model). Also, the emission probabilities are derived from the Bayesian net, which represents our training phase. Finally, ***delete*** states allow the model to pass through the gaps, existing in the Bayesian network to reach other emission states. These gaps are necessary in the model help prevent it from over-fitting of data as in fig 5 [28]. We use the forward algorithm to compute probabilities for each possible case recursively by reusing scores calculated for partial sequences using Eq. 1 to Eq. 3 respectively as thus:

$$F_j^M = Log\frac{eM_j(x_i)}{qx_i} + \log(aM_{j-1}M_j \exp\left(F_{j-1}^M(i-1)\right) + aI_{j-1}M_j \exp\left(F_{j-1}^I(i-1)\right) + aD_{j-1}M_j \exp\left(F_{j-1}^D(i-1)\right) \quad (1$$

)

$$F_j^I = Log\frac{eI_j(x_i)}{qx_i} + \log(aM_jI_j \exp\left(F_j^M(i-1)\right) + aI_jI_j \exp\left(F_j^I(i-1)\right) + aD_jI_j \exp\left(F_j^D(i-1)\right) \quad (2)$$

$$F_j^D = \log(aM_{j-1}D_j \exp\left(F_{j-1}^M(i)\right) + aI_{j-1}D_j \exp\left(F_{j-1}^I(i)\right) + aD_{j-1}D_j \exp\left(F_{j-1}^D(i)\right) \quad (3)$$

D. E. Benchmark Memetic Algorithm

Our framework is an adaptation of [30] that aims to evolve new malware from a known virus database. The first step is high-level of abstract representation (or genotype) of given virus that requires great understanding of virus functionality and structure. It determines quality of evolution achieved by proposed framework, while including functional details of the virus characteristics and that of the metamorphic engine in use. Some known feats/attributes are: date, application-to-infect, domain, port number, email attachment, mail-body, registry variable, file extension, process terminated, peer-to-peer propagation etc, which forms its abstract representation of the base virus to be taken as input into system (see fig 3).

Second step is the application of the evolutionary algorithm to the high-level representation. Thus, dataset is divided into: train (50%), cross validation (25%) and test (25%). The fitness of offspring as evaluated in Eq. 6 is a function of the similarity measure of the genes (chromosomes) with that of all stages of the framework. Individuals that evolved but do not match the training samples, their feats are stored and forms input to the next iteration. Thus, we have these conclusions as adopted by [30] thus: (1) new individuals are malware to be used during testing, whose abstract represented feats are fed-back into model, (2) new individual is an unknown Zmist virus, and (3) new individual is (not) Zmist virus.

In [29] facts 2 and 3 are established once executed within a sandbox in an operating system at real-time. Unlike [30] Generating Zmist virus for test as against having test data from base virus abstract representation, invalidates the experiment to some degree as mutation yields a better and fitter generation. Instead, we argued that a combination of the old feats and new feats as extracted from both dataset and metamorphic engine at each stage of the process as well as feedback into the system to generate newer variants to yield greater evolution (backward compatibility w.r.t. functionality to the base virus).

CGA initializes the hybrid with an entire population of 500-input (suitable abstract representation of base virus), computes individual fitness of each individual via Eq. 6 as well as selects 30-individual via tournament method to yield the new sub-pool (and determine individuals to proceed for mating). Selected data are moved for crossover and mutation so that model or network learns static/dynamic feats in the obtained data. With 30-individuals selected via tournament and 2-point crossover used, other parents contribute to yield new pool whose genetic makeup is a combination of both parents. Mutation will yield 3-random genes that are allocated new random value that still conforms to belief space. The number of mutation applied, depends on how far CGA is progressed (and how fit is the best fit individual in the pool). Thus, number of mutations equals fitness of the fittest individual divided by 2. New individuals replace old ones with low fitness values [28, 31, 42].

$$F = \sum_{i=1}^{k} \frac{f_i}{k} \quad (4)$$

[32-34]

Each particle in PSO (30-individuals from CGA) are moved over and encoded as a string of positions in multidimensional space. Position/Velocity updates are performed independently in each dimension (a merit of PSO). Though, not for such an evolution/permutation task, as candidate solutions depends on each other. Thus, two or more particles can have same value for velocity and position. Particles can take values outside the boundary after update, which breaks the rule of permutation. [35-40].

All conflicts are resolved using Eq. 5. With larger velocity, particles explore more space and will likely change (though all update formulas remain same). Velocity is limited to absolute values, which represents the difference between particles). This continues till an individual in the pool with a fitness of 0 is found. Thus, solution is reached [25-26]. Selection and mutation in GA ensures the first 3-beliefs are met; while velocity/position updates in PSO ensures that the fourth belief space is met, as time is of paramount interest. Also, influence function determines number of mutations takes place; And knowledge of how close task is to solution, has direct impact on how model is processed. Algorithm stops when best individual has fitness 0 [32].

F. Tradeoffs and Issues in Metamorphic Malware

Researchers designed routines to detect metamorphics (one-by-one) and detect varied sequences of code known to be used by given mutation engine via signature search. This method is proved inherently impractical, time-consuming and costly as each metamorphic requires its own detection program. Also, the mutation engine can seemingly randomly, generate billions variation of virus and different engines used by metamorphics make any identification somewhat unreliable. This has led to, mistakenly identifying one virus in place of another. Thus, our statistical model seeks to associate signature to metamorphic malware computed based on probability. Also, hybrid models are quite difficult to implement. But, we resolved the encoding via a structured learning which in turn, addresses the existing statistical dependencies amongst its variables to yield better pool via crossover/mutation. This feat can be adapted in areas of software evolution. This Genetic Algorithm trained Particle Swarm hybrid model combined with the Zmist metamorphic engine obfuscation yields Zmist variants in the shortest time that are highly independent, discrete and completely morphed copies of virus. Our resulting morphed copies were tested against normal files and against commercial virus scanners.

3. RESULTS AND FINDINGS

E. A. Result Findings and Discussion

To measure their effectiveness and classification accuracy, we adopt the misclassification rate of each model as well as its corresponding improvement percentages of the proposed model in comparison with those of other classification models for the diabetes data in both training and test data sets as summarized in Table 2 and Table 3, respectively. The equations for the misclassification rate and its improvement percentage of the unsupervised (B) model against those of the supervised (A) model, is respectively calculated as follows:

$$Misclassification\ Rate\ (MR) = \frac{No.\ of\ Incorrect\ Diagnosis}{No.\ of\ Sample\ set} \quad (5)$$

$$Improvement\ Percentage = \frac{MR(A) - MR(B)}{MR(A)} x100 \quad (6)$$

Table 2: Misclassification Rate of Each model

Model	Classification Errors	
	Training Data	*Testing Data*
PHMM	23.6%	39.2%
GAPSO	48.4%	53.7%

Table 3: Improvement Percentage

Model	Improvement %	
	Training Data	*Testing Data*
PHMM	56.03%	64.16%
GAPSO	42.79%	34.09%

Obtained results in tables 2 & 3, the proposed PHMM has a misclassification rate of 39.2% (resulting in false-positives and true-negatives error rate). Implying, it has a classification accuracy of about 60.9%; While, promising and/or shows an improvement of about 64.16%. In contrast, memetic algorithm (GAPSO) has a misclassification rate of 53.7% (resulting in false-positives and true-negatives error rate); while it promises to improve by 34.1%. Other parameter values of PSO led to slower or non-convergence. Generated variants viruses were tested against commercial antiviruses. Scanned with ESET, it was able to detect 56% of generated variants; while Norton Symantec detects 47% as in fig. 4a and 4b.

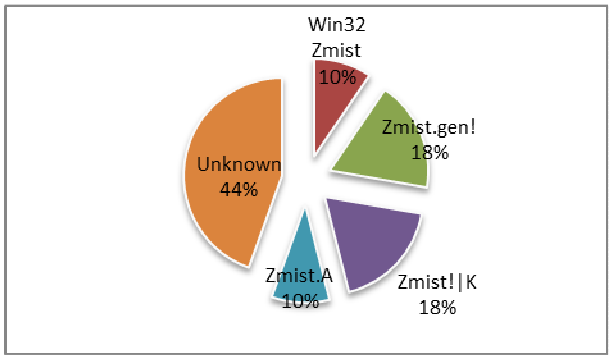

Fig. 4a: Evolved Variants from HMM scanned with Kaspersky

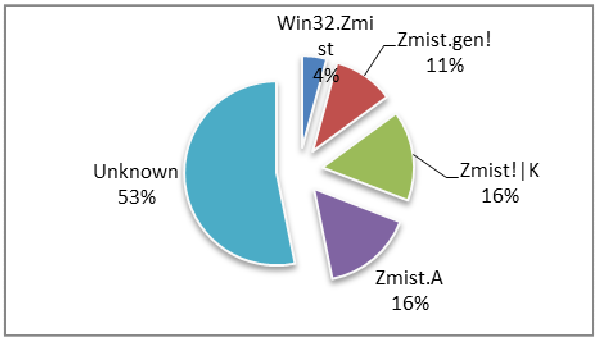

Fig. 4b: Evolved Variants via GAPSO scanned with Norton

Furthermore, metamorphic easily transform its codes as they propagate to avoid detection by using obfuscation methods to alters 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 antiviruses which are limited to static and dynamic analysis. Thus, they attack the following feats in a system: (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 analyze their own codes and must re-evaluate evolved or mutated codes generated (since complexity of transformation in the previous generation has a direct impact on its current state, how a virus analyses and transforms code in its current generation). Thus, they employ code conversion algorithm that helps them detect their own obfuscation and reordering (Ojugo, 2016).

4. CONCLUSION

In summary, with fitness function and selection common to both GA and PSO, we note that learning rates set between 0.2 and 0.35; and PSO feats set as: $\phi 1$, = 1.5, $\phi 2$ = 2.5, MaxGen = 500 epochs and ϖ = 0.14 yields better and faster convergence [32 – 40]. With [30], the proposed framework is posed as an evolvable system (though is adaptable to other domain tasks) is used in software evolution.

The process of evolution is associated with modifying an existent software or program with both backward cum forward compatibility, and also to emphasizes the concept of robustness and component reuse (Gray and Klefstad, 2005). Also, a wide variety of replicative and non-replicative malware can also be evolved via proposed framework to increase network security research and study.

REFERENCES

[1] Daoud, E and Jebril, I., "*Computer Virus Strategies and Detection Methods*", International Journal of Open Problems Computational Mathematics, 2008, Vol. 1, No. 2, [online]: www.emis.de/journals/IJOPCM/files/IJOPCM(vol.1.2.3.S.08).pdf

[2] Dawkins, R., "*The selfish gene*", Oxford University Press, Second Edition, 1989.

[3] Zakorzhevsky, E.R., "*Monthly malware statistics*", 2011, [online]:www.securelist.com/en/analysis/204792182/Monthly_Malware_Statistics_June_2011.

[4] Allenotor, D., "An Evolvable Framework for Metamorphics". Computing, Information Systems, Development Informatics and Allied Research Journal, 2016, Vol 7 No 2. Pp 33-40 Available online at www.cisdijournal.met

[5] Ojugo, A.A., "*Computer virus evolution: polymorphics analysis and detection*", Journal of Academic Research, 2010, Vol. 15, No. 8, p34 – 46.

[6] Ye, Y., Wang, D., Li, T and Ye, D., "*Intelligent malware detection based on association mining*", Journal of Computer Virology, 2008, Vol. 4, No. 4, p323–334, doi: 10.1007/s11416-008-0082-4.

[7] Szor, P., "*The Art of Computer Virus Research and Defense*", Addison Wesley Symantec Press. 2005, ISBN-10: 0321304543, New Jersey.

[8] Mishra, P., "*Taxonomy of software unique transformations*", 2003, www.cs.sjsu.edu/faculty/stamp/students/FinalReport.doc

[9] Orr, "*The viral Darwinism of W32.Evol: an in-depth analysis of a metamorphic engine*", 2006, [online]: available at http://www.antilife.org/files/Evol.pdf

[10] Orr, "*The molecular virology of Lexotan32: Metamorphism illustrated*", 2007, [online]: www.antilife.org/files/Lexo32.pdf

[11] Singhal, P and Raul, N., "*Malware detection module using machine learning algorithm to assist centralized security in enterprise network*", International Journal of Network Security and Applications, 2012, 4(1), doi: 10.5121/ijnsa.2012.4106, p61

[12] Rabek, J., Khazan, R., Lewandowski, S., Cunningham, R., "*Detection of injected, dynamic generated and obfuscated malicious code*", In Proceedings of ACM Workshop on Rapid Malcode, 2003, p76.

[13] Filiol, E., "*Computer Viruses: from Theory to Applications*", New York, Springer, 2005, ISBN 10: 2287-23939-1.

[14] Hashemi,S., Yang, Y., Zabihzadeh, D and Kangavari, M., "*Detecting intrusion transactions in databases using data item dependencies and anomaly analysis*", Expert Systems, 2008, Vol. 25, No. 5, p460, doi:10.1111/j.1468-0394.2008.00467.x

[15] Grimes, R., "*Malicious Mobile Code: Virus Protection for Windows*", O'Reilly and Associates, Inc., Sebastopol, CA, USA, 2001.

[16] Cohen, F., "*Computer viruses: theory and experiments*", Computer Security, 1987, 6(1), p22-35.

[17] Sung, A., Xu, J., Chavez, P., Mukkamala, S., "*Static analyzer of vicious executables*", Proceedings of 20th Annual Computer Security Applications Conference, IEEE Computer Society, 2004, p326-334.

[18] Venkatesan, A., "*Code obfuscation and metamorphic Virus Detection*", Master thesis, San Jose State University, 2006, www.cs.sjsu.edu/faculty/students/ashwini_venkatesan_cs298report.doc

[19] Konstantinou, E., "*Metamorphic virus: analysis and detection*", Technical report (RHUL-MA-2008-02), Dept. of Mathematics, Royal Holloway, University of London, 2008.

[20] Walenstein, R., Mathur, M., Chouchane R., and Lakhotia, A., "*The design space of metamorphic malware*", In Proceedings of 2nd Int. Conference on Information Warfare, 2007, p243.

[21] Wong, W., "*Analysis and Detection of Metamorphic Computer Viruses*", Master's thesis, San Jose State University, 2006, http://www.cs.sjsu.edu/faculty/students/Report.pdf

[22] Borello, J and Me, L., "*Code obfuscation techniques for Metamorphics*, 2008, [online]: available at www.springerlink.com/content/233883w3r2652537
[23] Aycock, J., "*Computer Viruses and malware"*, Springer Science and Business Media, 2006.
[24] VX Heavens Virus Collection, [online]: http://vx.netlux.org/
[25] Ojugo, A.A and Yoro, R.E., "*Computational intelligence in stochastic solution for Toroidal Queen",* Progress in Intelligence Computing Applications, 2013a, Vol. 2, No. 1, doi: 10.4156/pica.vol2.issue1.4, p46
[26] Ojugo, A.A., Emudianughe, J., Yoro, R.E., Okonta, E.O and Eboka, A.O., "*Hybrid artificial neural network gravitational search algorithm for rainfall runoff",* Progress in Intelligence Computing and Applications, 2013b, Vol. 2, No. 1, doi: 10.4156/pica.vol2.issue1.2, p22.
[27] Ojugo, A.A., Oyemade, D.A., Allenotor, D., Longe, O.B and Anujeonye, C.N., "*Comparative Stochastic Study for Credit-Card Fraud Detection Models,*. African Journal of Computing and ICT, 2015, Vol 8, No. 1, Issue 2. Pp 15-24.
[28] Ojugo, A.A., "*A profile hidden markov model for forecasting energy spread options direction and volatility,* Technical Report for Dynamic High Performance Computing Research Group of the Federal University of Petroleum Resources Effurun, 2013, FUPRE-TR-DHCP-08, Pp 10-24.
[29] Ramage, D., "*Hidden markov model fundamentals",* Lecture notes in Computer Science, [online source]: www.springerlink.com www.springerlink.com/content/lecturen_notes/cs/235483 w3r2652537
[30] Noreen, S., Ashraf, J and Svrenahak, K., "*Malware detection using evolutionary models"*, International Journal of Virology, 2008, Vol. 23, No. 2, p123-132.
[31] Ojugo, A., A.O. Eboka., E.O. Okonta., E.R. Yoro and F.O. Aghware., "*Genetic algorithm trained rule-based intrusion detection system"*, Journal of Emerging Trends in Computing and Information Systems, Vol. 3, No. 8, 2012, Pp 1182-1194
[32] Ursem, R., Krink, T., Jensen, M.and Michalewicz, Z., "*Analysis and modeling of controls in dynamic systems",* IEEE Transaction on Evolutionary Computing, 2002, 6(4), p378-389.
[33] Clerc, M., "*The .Aswarm and the queen: towards a deterministic and adaptive particle swarm optimization"*, In Proceedings of Evolutionary Computation (IEEE), 1999, 5, p123-132.
[34] Gray, J and Klefstad, R., "*Adaptive and evolvable software systems: techniques, tools and applications"*, 38th Annual Hawaii Int. Conf. on System Sciences, 2005, p274, IEEE Press.
[35] Hassan, R and Crosswley, W., "*Variable population-based sampling for probabilistic design optimization and with a genetic algorithm"*, Proceedings of 42nd Aerospace Science, p32, Reno: NV, 2004.
[36] Hassan, R., Cohanin, B., De Wec and Venter, G., "*Comparison of particle swarm optimization and genetic algorithm",* In Proceeding of 44th Aerospace Science, 2004, Washington, p56.
[37] Homaifar, A.A., Turner, J and Ali, S., "*N-queens problem and genetic algorithms"*, In Proceedings of the IEEE Southeast conference, 1992, p262.
[38] Hu, X., Eberhart, R.C and Kennedy, J., "*Solving constrained nonlinear optimization problems with PSO,* In Proceedings of the Multi-conference on Systems, Cybernetics and Informatics, 2005a, p234.
[39] Hu, X., Eberhart, R.C and Shi, Y., "*Swarm intelligence for permutation optimization: study of n-queens"*, Proceedings of IEEE Genetic Evolutionary Computing on Memetic Algorithm, 2005b, p243
[40] Kennedy, J and Mendes, R., "*Population structure and particle swarm performance"*, In Proceedings of the IEEE Congress on Evolutionary Computation, 2002, p-1671, Honolulu
[41] Lakhotia, A., Kapoor, A and Kumar, E.U., "*Are metamorphic computer viruses really invisible*?", 2004, Part 1, Virus bulletin, p5-7.
[42] Reynolds, R., "*An introduction to cultural algorithms"*, IEEE Transaction on Evolutionary Programming, 1994, p131.