

B.Sc. in Computer Science and Engineering Thesis

# **Improved Technique for Polymorphic Worm Detection and Multiple String Alignment using Genetic Algorithm**

Submitted by

Fahima Khanam

201314050

K. M. Salman Soumik

201314051

Jubair Hossain

201314057

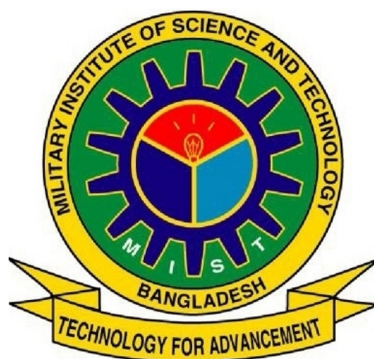
Supervised by

Professor Dr. Md. Mahbubur Rahman

Lecturer Wali Mohammad Abdullah

Military Institute of Science and Technology

Mirpur Cantonment, Dhaka



**Department of Computer Science and Engineering  
Military Institute of Science and Technology**

January 2017

# CERTIFICATION

This thesis paper titled “**Improved Technique for Polymorphic Worm Detection and Multiple String Alignment using Genetic Algorithm**”, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in January 2017.

## **Group Members:**

**Fahima Khanam**

**K. M. Salman Soumik**

**Jubair Hossain**

## **Supervisor:**

---

Md. Mahbubur Rahman  
Professor  
Department of CSE  
Military Institute of Science and Technology  
Mirpur Cantonment, Dhaka.

---

Wali Mohammad Abdullah  
Lecturer  
Department of CSE  
Military Institute of Science and Technology  
Mirpur Cantonment, Dhaka.

# CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis paper, titled, “Improved Technique for Polymorphic Worm Detection and Multiple String Alignment using Genetic Algorithm”, is the outcome of the investigation and research carried out by the following students under the supervision of Professor Dr. Md. Mahbubur Rahman, Lecturer Wali Mohammad Abdullah, Military Institute of Science and Technology, Mirpur Cantonment, Dhaka.

It is also declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

---

Fahima Khanam  
201314050

---

K. M. Salman Soumik  
201314051

---

Jubair Hossain  
201314057

# ACKNOWLEDGEMENT

We are thankful to Almighty Allah for His blessings for the successful completion of our thesis. Our heartiest gratitude, profound indebtedness and deep respect go to our supervisors, Professor Dr. Md. Mahbubur Rahman, Lecturer Wali Mohammad Abdullah, Military Institute of Science and Technology, Mirpur Cantonment, Dhaka, for their constant supervision, affectionate guidance and great encouragement and motivation. Their keen interest on the topic and valuable advices throughout the study were of great help in completing thesis.

We are especially grateful to the Department of Computer Science and Engineering (CSE) of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, we would like to thank our families and our course mates for their appreciable assistance, patience and suggestions during the course of our thesis.

Dhaka  
January 2017

Fahima Khanam

K. M. Salman Soumik

Jubair Hossain

# ABSTRACT

This paper aims to propose an improved technique for generating accurate signature to detect polymorphic worm efficiently. In this paper, we present a method to generate signature that can be used with Simplified Regular Expression mechanism. There are three types of mechanisms for representing and detecting polymorphic worm codes, as Autograph, Polygraph and Simplified Regular Expression (SRE). Because it preserves all the essential features of polymorphic worm- wildcard strings and distance restrictions. Existing signature detection methods use sequence alignment to generate the signature, as a result many essential invariant parts of polymorphic worm codes are lost in the process. Our proposed method uses longest common substring algorithm to generate all common substrings and stores them so that it can produce accurate signature without loss of essential features. Experimental results of our work show that our technique is accurate enough after comparing it with two existing algorithms. We also research in the field of Bio-informatics by manipulating Sequence Alignment technique. In molecular sequence analysis Multiple Sequence Alignment(MSA) plays a vital role as it calculate the maximize similarities between two or more sequences. One of the technique for solving Multiple Sequence Alignment(MSA) is genetic algorithm. Main advantage of solving Multiple Sequence Alignment(MSA) by genetic algorithm is, the only thing that we must concern about is the fitness function. In this paper a modified edit distance matrix in mutation operator in genetic algorithm is proposed for solving multiple sequence alignment problems. The main modification is done in edit distance matrix where we have used swapping operations instead of Insertion or Deletion(INDEL) operations. Unweighted Pair-Group Method with Arithmetic mean(UPGMA) method is used to generate the guide tree where our modified edit distance matrix has been used which gives better or equal performance than previous method.

# TABLE OF CONTENT

<i>CERTIFICATION</i>	ii
<i>CANDIDATES' DECLARATION</i>	iii
<i>ACKNOWLEDGEMENT</i>	iv
<i>ABSTRACT</i>	1
<b>List of Figures</b>	<b>5</b>
<b>List of Tables</b>	<b>6</b>
<b>List of Abbreviation</b>	<b>8</b>
<b>List of Symbols</b>	<b>9</b>
<b>1 Introduction</b>	<b>10</b>
1.1 Introduction . . . . .	10
1.2 Motivations . . . . .	11
1.3 Objectives . . . . .	12
1.3.1 Polymorphic Worm Signature Detection . . . . .	13
1.3.2 Improving MSA Algorithm . . . . .	13
<b>2 Preliminaries</b>	<b>14</b>
<b>3 Literature review</b>	<b>20</b>
3.1 Introduction . . . . .	20
3.2 Polymorphic Worm Detection Technique . . . . .	20
3.3 Needleman-Wunsch Algorithm . . . . .	23
3.4 Smith-Waterman Algorithm . . . . .	24

3.5	Tree-Base Algorithm . . . . .	24
3.6	Yadav and Banka's Algorithm . . . . .	25
3.7	CSR (Contiguous Substrings Rewarded) algorithm . . . . .	26
3.8	M-STR-EC-LCS Algorithm . . . . .	27
3.9	Beam Search Algorithm . . . . .	27
3.10	ECSR (Enhanced CSR) Algorithm . . . . .	28
<b>4</b>	<b>Proposed Method</b>	<b>30</b>
4.1	Polymorphic Worm Signature Detection . . . . .	30
4.1.1	Proposed Algorithm . . . . .	30
4.1.2	Time & Space Complexity . . . . .	31
4.1.3	Simulation of Proposed Algorithm . . . . .	31
4.2	An Improved Genetic Algorithm for MSA Problem . . . . .	32
4.2.1	Crossover . . . . .	32
4.2.2	Mutation . . . . .	33
<b>5</b>	<b>Experiments and Results</b>	<b>37</b>
5.1	Polymorphic Worm Signature Detection . . . . .	37
5.1.1	Introduction . . . . .	37
5.1.2	Experimental Result . . . . .	38
5.1.3	Performance Analysis . . . . .	39
<b>6</b>	<b>Conclusion</b>	<b>42</b>
6.1	Conclusion . . . . .	42
6.2	Future Work . . . . .	42
	<b>References</b>	<b>42</b>
<b>A</b>	<b>Algorithms</b>	<b>46</b>
A.1	Polymorphic Worm Signature Detection Algorithm . . . . .	46

<b>B</b>	<b>Codes</b>	<b>48</b>
B.1	Polymorphic worm detection code . . . . .	48
B.2	Random test case generator code . . . . .	51



# LIST OF FIGURES

2.1	Polymorphic Worm structure . . . . .	16
2.2	Red Code II Structure . . . . .	16
2.3	Example of a Multiple Sequence Alignment. . . . .	17
2.4	Example of Global Alignment. . . . .	17
2.5	Example of Local Alignment. . . . .	18
3.1	Sequence Alignment Result . . . . .	24
3.2	Sequence Alignment Result . . . . .	27
3.3	Keyword Trees . . . . .	28
3.4	Enhanced CSR Result . . . . .	29
4.1	Parent chromosome . . . . .	32
4.2	Multipoint crossover . . . . .	32
4.3	Child B chromosome . . . . .	33
4.4	Child A chromosome . . . . .	33
4.5	Converting CTGA string to GTAC . . . . .	34
4.6	Converting CGTA string to CTGA . . . . .	34
4.7	Converting CGTA string to GTAC . . . . .	35
4.8	Combine sequence of all strings . . . . .	36
5.1	Scoring Comparison Chart . . . . .	40
5.2	Accuracy Comparison Chart . . . . .	40

## LIST OF TABLES

4.1	Performance table of Edit distance matrix . . . . .	35
5.1	Generated Signatures . . . . .	38
5.2	Scoring Result Comparison . . . . .	39
5.3	Accuracy Percentage Comparison . . . . .	40

# List of Algorithms

1	All Common Substring Search . . . . .	46
2	Multiple String Common Substring . . . . .	47

## LIST OF ABBREVIATION

- CSR** : Contiguous Sub-string Rewarded  
**ECSR** : Enhanced Contiguous Sub-string Rewarded  
**MSA** : Multiple Sequence Alignment  
**DP** : Dynamic Programming  
**DNA** : Deoxyribo Nucleic Acid  
**RNA** : Ribo Nucleic Acid  
**GA** : Genome Alignment  
**INDEL** : Insertion or Deletion  
**M-STR-EC-LCS** : Multiple String Excluding Constraints Longest Common Sub-sequence  
**UPGMA** : Unweighted Pair-Group Method with Arithmetic mean  
**IP** : Internet Protocol  
**DMZ** : Demilitarized Zone  
**SRE** : Simplified Regular Expression  
**NSG** : Network based Signature Generation

## LIST OF SYMBOLS

$k_m$	: Number of matches
$W_m$	: Weight for matches
$k_d$	: Number of mismatches
$W_d$	: Weight for mismatches
$k_g$	: Number of gaps
$\delta$	: Weight for gaps

# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction

Security is the level of impenetrable to or assurance from damage. It applies to any profitable expedient for instance, personal assets or associations. It is commonly identified with the alleviation of threats to cherished values. Security perception is poorly represented to objective security.

Bioinformatics is an emerging discipline of Biotechnology which has now become the heart of modern biological research. It applies “informatics techniques” to understand and organize the information associated with the molecules, on large scale. It is an indispensable ally of researchers. It involves discovery, development and implementation of computational algorithms and software tools that helps to understand the biological processes with the goal to serve the humankind. Computers are used to gather, store, analyze and integrate biological and genetic information which can then be applied to gene-based drug discovery and development. Its application is not only limited to biomedical, pharmaceutical and agriculture but some of its recent applications even extends to bio-diversity conservation, bio-polymers and even space science. One of the most important parts of bioinformatics is sequence alignment.

There are three basic aims of bioinformatics. First, biological data are organized in form of a database. This allows the researchers an easy access to existing information and submit new entries. These data must be annotated to give a suitable meaning or to assign its functional characteristics. The database must also be able to correlate between different hierarchical of information. For example: GenBank for nucleotide and protein sequence information, Protein Data Bank for 3D macromolecular structures. The second aim is to develop tools and resources that aid in the analysis of data. For example: BLAST to find out similar nucleotide/amino-acid sequences, ClustalW to align two or more nucleotide/amino-acid sequences, Primer3 to design primers probes for PCR techniques etc. The third and the most important aim of bioinformatics is to exploit these computational tools to analyze the biological data interpret the results in a biologically meaningful manner. In Bioinformatics, we can now conduct global analyzes of all the available data with the aim of uncovering principals that apply across many systems and highlight novel features [25].

In biological aspects Multiple Sequence Alignment (MSA) plays the most pivotal role for inspecting biological properties. The main goal of MSA is to generate an incisive, information rich summary of sequence data. When sequences alignment is done between three or more biological sequences such as Protein, DNA or RNA (Auyeung and Melcher, 2005; Wei et al., 2013) then it is known as multiple sequences alignment (Hamidi et al., 2013). For reviling the relationship between collections of evolutionarily or structurally related protein one of the standard techniques in bioinformatics is sequence alignment. Sequence alignment are widely used for improving the secondary and tertiary structure of protein and RNA sequences, which is used for drug designing and also to find distance between organism. In MSA, the foremost effort is to find the optimal alignment for a group of biological sequences. The main objective of an MSA is to align sequences which can show the biological relationship between the input sequences. Developing a reliable MSA program is not easy.

## 1.2 Motivations

We have studied the existing methods of detecting polymorphic worm and analysis the characteristics. One of the major characteristics of polymorphic worm is its unpredictability. Polymorphic shell code always try to confuse the detection system with change in pattern. So it's not certain whether the longest common substrings will be the invariant part of the worm, it may be a garbage. So the probability of the worm signature being a garbage can be reduced when there is a third stream of worm. If the longest ones are invariant in all three streams, it's all right, but when it's not then algorithm by Tang et al. fails to generate any output signature as it has already removed all the other shorter substrings from the signature. Algorithm by Aljawarneh et al. can generate the signature, but it needs to find all the common substrings between the latter two strings again, which takes time. Also there is a case when both sequences have two or more common substrings of same length, but not in same sequence. So this case creates a confusion regarding which common substring to be taken as signature. For example:-

Seq1: oxnxexzxtwox

Seq2: ytwoyoynyeyz

Seq3: oznzez

For Seq1 and Seq2, both methods by Tang et al. and Aljawarneh generate “two” as worm signature, as it is the longest. But when comes seq3, we can see that the longest ones previously determined is not in the signature. Rather the signature is actually “o”, “n”, “e”, “z”. But Tang et al. only kept the signature “two” from previous, so it can't match anything. Whereas Aljawarneh et al. can generate the correct result, but it needs to find all the common substrings again. Another example is:

Seq1: onetwox

Seq2: tnoonex

Seq3: yytwo

In the above example, both the algorithms by Tang et al. and Aljawarneh et al. results in “one” for first two streams. But both “one” and “two” are of same length and common in both strings, but have different sequences. So it’s not clear which one is the actual signature. It becomes clear after Seq3, as “two” is the accurate signature. But both the algorithms discarded “two” in 1st step. So there must be a mechanism to keep all the candidates of worm signature.

For improving the existing system we introduce the longest common substring along with stored substrings which stores substrings in each iteration for multiple infected network flow sequences. We follow dynamic programming technique to improve the computation efficiency. At the very last, accurate signature are generated which is solely better than other algorithms.

MSA is now become the heart of bioinformatics for solving multiple DNA, RNA sequences. There are several methods to solve MSA problems such as iterative, classical and progressive. All these algorithm are based on global or local alignment. These two methods have different properties. Both of them have some advantage and disadvantage. Local alignment is preferable but finding the regions of similarity is not easy. Needleman and Wunsch algorithm is based on global alignment technique whereas Smith-Waterman algorithm used local alignment technique. Both of them used dynamic programming(DP). We can solve MSA problem using DP but only when the number of sequences is two. But as MSA is a combinational problem (NP hard), when the number of sequences increases we can not solve the problem using DP [29]. To overcome such situations researchers switch to iterative or stochastic process. So we have worked with genetic algorithm. The main advantage of using GA for MSA problem is that it does not requires any particular source of algorithm to solve a given problem. Only thing that is needed is fitness function. As the research is based on sequences of larger length therefore GA is considered over DP.

Genetic algorithm is more powerful than any other technique for solving MSA problem. In Yadav and Banka’s method they have used INDEL edit distance matrix in mutation operator. Our main target was solving the MSA problem with genetic algorithm using guide tree with minimum edit distance operations.

### 1.3 Objectives

Our thesis is on the proposal of an enhanced technique for detecting the polymorphic worm code from the network flow which are considered as input sequences of the algorithm and



also for solving MSA problem by genetic algorithm with slight change in edit distance matrix. Many algorithm and methods exist to solve the problem but not enough accurate to recognize the infected part.

### **1.3.1 Polymorphic Worm Signature Detection**

Our goals for polymorphic worm signature detection are:

- (a) Introducing anatomy of polymorphic worm.
- (b) Make efficient of the existing method.
- (c) Improve scoring function result quality.
- (d) Gain accurate polymorphic worm signature in a consistent way to identify the worm sequences.
- (e) Comparing the outcome of our contributed algorithm among the existing methods.
- (f) Comparing the outcome of scoring function among the existing methods.
- (g) Plotting the data of the output and scoring function result in graph to represent the graphical importance of our technique..
- (h) Summarize the limitation of our approached method.

### **1.3.2 Improving MSA Algorithm**

Our goals for the MSA problem are:

- (a) Manually solving the MSA problem with randomly taken three to four multiple sequences.
- (b) Making comparison between the proposed method and the existing method.

## CHAPTER 2

# PRELIMINARIES

This section provides a detail idea about the basic concept of the related terms used in the paper such as Worms, Polymorphic worms, Structure of worms, Alignment, Sequence Alignment, Multiple Sequence Alignment, BAliBase, UPGMA, Global Alignment, Local Alignment and Edit Distance Matrix.

### **Worm**

Worm is a type of software program that is considered as malicious and harmful for the system which reproduces itself simultaneously in order to spread to other computers. Network communication is the best way to attack in any host computer for this malware but it also depends on the system security weakness. Unlike computer virus, it has no necessity to attach itself to a computer program. It mostly consumes network bandwidth whereas virus destroys data and information in a host system [1,2].

Worms are generally detected by content-based filtering using worm signature. The signature of a worm contains the characteristics and features of the worm. There are two types of signatures- exploit based and vulnerability based. We generate exploit based signature as each polymorphic virus has its distinct exploit characteristics [3].

### **Types of Worms**

#### **I EMAIL WORMS**

An email worms spread itself by using a host client of email service. It often infects a computer by initializing itself by sending a link toward the host computer within an email. When user click on that link by impulsion of the email service, the worm unintentionally is installed in the user PC and start infecting the system and drive files accordingly by thieving the precious and secret information of the user. Not only one way it can infect the PC, it will send an attachment by emailing the host by that and when opened the email, it will start the infection. Some users try to start these attachments thinking that may be these files are multimedia files. Once the worm is installed, it has the next to search the host computer for no particular email addresses contained on it. Without users knowledge, it starts the process again to send the worm without any input from the host source. In 2000, a worm named “ILOVEYOU” compromised the system of millions of computers worldwide.

## II INTERNET WORMS

Internet worms are basically independent programs. The main strategy of their spreading through the internet using a host machine and continue to scan inter-connected network systems for any kind of vulnerable machines. After locating a security weak machine in the network, the worm will infect it and begin the process to invade the appointed system again. Internet worms are often created to exploit recently discovered security issues on machines that haven't installed the latest operating-system and security updates.

## III FILE-SHARING NETWORKS WORMS

File-sharing worms take advantage of the fact that file-sharers do not know exactly what they are downloading. The worm will copy itself into a shared folder with an unassuming name. When another user on the network downloads files from the shared folder, they will unwittingly download the worm, which then copies itself and repeats the process. In 2004, a worm called "Phatbot" infected millions of computers in this way, and had the ability to steal personal information, including credit card details, and send spam on an unprecedented scale.

## IV INSTANT MESSAGE AND CHAT ROOM WORMS

These work in a similar way to email worms. The infected worm will use the contact list of the user's chat-room profile or instant-message program to send links to infected websites. These are not as effective as email worms as the recipient needs to accept the message and click the link. They tend to effect only the users of the particular program.

### Polymorphic worm

Polymorphic worms can modify their program code while replicating themselves, which lets them to infiltrate operating systems and software. They can damage networks by consuming bandwidth and can compromise important private information. [4, 7, 8] According to Xiao et al. [9] worm spreads in three steps:

- **Victim selection:** The worm targets next IP address as victim.
- **Shifting worm:** Worm sends a copy of itself to the target victim.
- **Exploitation:** After reaching the new victim device, the worm executes itself.

### Polymorphic Worm Creation

Polymorphic worms are created in such a way that at each mutation it generates a unique copy of itself without changing its functionality so that it can confuse worm detection techniques by disguising its payload using encoding techniques which consists of Decryption Routine, Decryption key, Encrypted Worm Code and Exploit Code [8] shown in Figure 2.1.

Decryption Routine	Decryption Key	Encrypted Worm Code	Exploit Code
-----------------------	-------------------	------------------------	-----------------

Figure 2.1: Polymorphic Worm structure

Some common Polymorphic shell code generators are ADMmutate, Clet and TAPiON which can comprise Garbage, equivalent code and encoding within polymorphic shell code . Polymorphic worm samples are encrypted with different encryption and decryption keys where the included decryption keys differ for each copies of worm, though the Decryption routine and Exploit code models remain equivalent. The whole worm is not encrypted, rather a portion of it remains un-encrypted so that t can exploit new victims and run the decryption routine to execute the invariant payload.

### Structure of Polymorphic Worm

Polymorphic worm mutates its byte sequence at each of its exploitation. So there are two parts within a polymorphic worm code- the invariant bytes and wildcard bytes. The invariant bytes are fixed, it doesn't change at replication, so that it can successfully complete the exploit. Wild card bytes are changed at each mutation for different worm samples. In Fig 2 Tang et al. [6] explains the polymorphic worm structure. In Code Red II is such a polymorphic worm that has seven invariant contents: "GET", ".ida?", "XX", "%u", "%u7801", "=" and "HTTP/1.0\r\n". Among these invariant parts, "%u7801" is 4 bytes after "%u" and "=" is seven bytes before "HTTP/1.0\r\n" which defines the distance restriction that is the "possible beginning of a substring" or "number of characters between two substring" as distance restriction is important for worm exploitation and it is crucial for implementation for successful detection.



Figure 2.2: Red Code II Structure

In fact, the existent approaches to detect Red Code II are not fully perfect in terms of giving the distance restriction between the invariant substrings. There are also some limitation those are found out as the substrings must not be length of 1 or it is too complex to extract the one length possible substrings. And also no fixed regions can be selected to get the invariants contents.

### Alignment

The arrangement of two or more biological sequences in such a way that tells us at what point the sequences are similar and at what point they differ is known as alignment. An

alignment is said to be the optimal one, if it has more similar sequences as compared to dissimilar sequences.

### Sequence Alignment

It is a way of arranging DNA, RNA or protein in sequences to identify regions of similarity which helps to infer structural, functional or evolutionary relationship between sequences. It aims to infer clues about the unknown sequence by inferring biological characteristics of the matched sequence. One of the most challenging tasks in sequence alignment is its repetitive and time-consuming alignment matrix computations.

### Multiple Sequence Alignment

It is a sequence alignment of three or more biological sequences such as DNA, RNA and protein. An MSA can be obtained by inserting gaps “-” at proper places such that no column in the sequences contains only gap character. Insertion of gaps will result in equal length sequences in the resulting alignment [32].

Example:

<i>Sequence1</i>	<b>-TCAGGA-TGAAC----</b>
<i>Sequence2</i>	<b>ATCACGA-TGAACC----</b>
<i>Sequence3</i>	<b>ATCAGGAATGAATCC--</b>
<i>Sequence4</i>	<b>-TCACGATTGAATCGC-</b>
<i>Sequence5</i>	<b>-TCAGGAATGAATCGCM</b>

Figure 2.3: Example of a Multiple Sequence Alignment.

### Global Alignment

Global alignment is the procedure where end to end alignment of the entire sequence is done. It contains all letters from both the query and target sequences. When two sequences have approximately the same length and are quite similar, they are suitable for global alignment. Global alignment are usually done for comparing homologous genes like comparing two genes with same function or comparing two proteins with similar function. An example of global alignment technique is the Needleman-Wunsch algorithm.

Example:

1	2	3	4	5	6	7	8	9	10	11
C	G	T	C	C	G	A	A	G	T	G
			.							
★	★	T	A	C	G	A	A	★	★	★

Figure 2.4: Example of Global Alignment.

### Local Alignment

Local alignment is a procedure where local regions with the highest level of similarity is

found between two sequences. A local alignment aligns a substring of the query sequence to a substring of the target sequences. Suitable for aligning more divergent sequences or distantly related sequences. It is used for finding out conserved patterns in DNA sequences or conserved domains or motifs in two proteins. In local alignment we have to find the region of similarity first and then aligned this part of sequences [30]. A general local alignment method is Smith-Waterman algorithm.

Example:

	3	4	5	6	7	8
	T	C	C	G	A	A
		-				
T	A	C	G	A	A	

Figure 2.5: Example of Local Alignment.

### Edit Distance

Edit distance is a process of measuring how dissimilar two strings from each other by counting the minimum number of operations required to transform one into other sequences.

### UPGMA

The unweighted pair-group method with arithmetic mean(UPGMA) is the simplest method for constructing trees. It is an agglomerative hierarchical clustering method .

### Genetic Algorithm

Genetic Algorithm is a type of iterative algorithms which allows an efficient and robust search. In the search process, a genetic algorithm starts with an initial state (population) in the solution space and in every search step, it produces a new and usually a better set of solutions. At each stage, GA moves forward towards producing a better solution which may led to minimize the change of getting trapped into a local extrema (Michalewicz, 1992) [32].

### BAlibase

BAlibase dataset is considered to be the standard dataset for alignment of protein sequences. It consists of variable lengths protein sequences which includes 218 sets of sequences taken from different sources. Here, the sequences are differentiated based on their similarity and structure in PDB database (Neshich et al; 1998). To evaluate the quality of the obtained alignment, the BAlibase defined two sets of score namely SP Score and TC Score [32].

### Fitness Function

The fitness function simply defined is a function which takes a candidate solution to the problem as input and produces as output how “fit” or how “good” the solution is with respect to the problem in consideration. Calculation of fitness value is done repeatedly in a GA and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow. A fitness function should possess the following

characteristics :

- (a) The fitness function should be sufficiently fast to compute.
- (b) It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

# CHAPTER 3

## LITERATURE REVIEW

### 3.1 Introduction

Polymorphic worm can be determined by matching invariant sub-strings which contain all different characteristics of the worm in the network flow. The detection method can be classified into three classes, content based detection, behavior based detection, and pattern based detection. Content based detection is done by producing such information from worm content that matches the worm [10–14]. Behavior based detection monitors the system and network activities of each file and when any anomaly is found it is detected as worm [15–17]. Pattern based detection is like content based detection It matches strings to find a pattern between two copies of same worm so that it can detect a new worm using that pattern we are going to discuss about pattern based worm detection techniques as we approach to detect the worm through those techniques.

There are three types of pattern based detection to identify polymorphic worm in the network flow

- Autograph
- Polygraph
- Simplified Regular Expression

### 3.2 Polymorphic Worm Detection Technique

#### Autograph

Autograph is a preliminary approach based on automatically generate worm signatures for Internet worm. In this approach high sensitivity and high specificity are expressed using filtering based on network flow contents. Kim et al. [18] give an analysis to that kind of worms those are disseminated over TCP transport. This type of signatures consists of three portions, protocol number of IP address, targeted victim port number and also byte sequence. Payload of the worm is detected when the sequences of the worm is matched in follow of the same IP address with the same destination port.



Autographing repeat the process for the majority of prevalent content block is chosen as signature and then all flows found in this content block are eliminated. This process is repetitive, with remaining flows continuing until the fraction of the entire flows in the pool has been enclosed. Autograph will report the set of chosen signatures in Bros signature feature at the end of the process. The quality of the signature generated by this system depends upon the size of the content block. When the size of the content block has been made small, it is in most cases autograph signature generating. Then, the suspicious flow after that passes through the generator signatures, which automatically generates various classes of signatures.

For executing this techniques, Demilitarized Zone(DMZ) system is used to give the inputs as in the network flow contents. For detecting the worm from the huge given input to the Autograph Monitor two steps are followed, suspicious flow selection and signature generation. Suspicious flow divides the sequences of the contents in which the most number of common byte sequences are there for achieving the priority of the content block based on a higher count and considered as the probable signature of the detected polymorphic worm [19].

### **Polygraph**

Polygraph is based on pattern detection as it is used its technique when multiple strings involve as input strings whereas autograph is based on only single string. As there should have multiple strings to detect the signature of polymorphic worm, the probability of the output is much accurate than autograph. But both the technique are on the payload of the suspicious polymorphic worm strings. According to Newsome et al. [20] a low false positive and a low false negative is calculated for generating signature from substrings and stratify into three categories:

(a) **Conjunction signature:**

A signature, which consists of a set of tokens, and the signature match a payload when entire tokens in the set are found in it unordered. This kind of signature matches the multiple invariant tokens provided in a polymorphic worm payload. Here, matching multiple tokens is more exact than matching one of these tokens only.

(b) **Token-subsequence signature:**

A signature that consists of an ordered set of tokens. A stream matches a token-subsequence signature when the flow has the same sequence of tokens in the signature with identical ordering. Signatures of this kind are simply expressed as regular expressions and allowed to be used in current IDSs. For the equivalent set of tokens, a token sub sequence signature is more precise than a conjunction signature due to the presence of order constraints in a token sub sequence signature.

(c) **Bayes signature:**

Bayes signature distinguishes a worm from an innocuous stream by probabilistic matching, distinct from conjunction and token sub sequence signature, which are based on the exact prototype match. Signatures consist of a set of tokens, which are each associated with a score and overall threshold. It provides probabilistic matching for a given flow by computing the probability that the flow is a worm by the scores of the tokens provided in the flow. When the probability of the result is more, the threshold of the flow is classified as a worm.

Conjunction signature has some tokens by which it can check every worm payload and declared as the polymorphic worm when all the tokens are found un-orderly in the input strings. As multiple strings are used so the result is better than autograph.

Identical ordering among tokens is the main concept of Token-subsequence signature. As ordering is considered among the multiple strings then results become accurate than conjunction signature.

Every tokens in the signature generated by Bayes Signature has scoring and threshold which can be expressed as the tolerance range to make decision of the flow as a worm. It learns from affected pool and include tokens from unassociated worms.

**Simplified Regular Expression**

Simplified Regular Expression (SRE) develops such a strategy to make accurate signatures of polymorphic worm from multiple sequences by different alignment algorithms in form of regular expression though consists of limited syntax rules whereas Autograph and Polygraph fail to produce perfect and accurate result. Autograph proves that it is not effective and efficient way to extract the worm sequence meanwhile polygraph can lose part of information considered as important to express the full characteristics of the information as it generates token-based signature also known as byte sequence [6]. On the other hand, the autograph and polygraph technique fail to trace the perfect distance restriction as number of character among the tokens where SRE can produce better distance information that is more accurate for detecting the invariant worm sequence [21]. The authors have used three qualifiers to express the SRE signature of polymorphic worm sequence, “\*”, “[k1,K2]” and “[k]”. Instead of “\*” they use “.\*” along with zero length string. Expression illustration can be written like this expression as “three”[2]“five”[3,4]\*“four” and it contains invariant parts of the sequence [6]. SRE method is a modified approach of pairwise alignment as it is based on multiple strings of probable infected file.

This technique verifies that for all their experiments on the several approaches of detection of polymorphic worm signatures, compared with present network-based signature generation systems (NSGs) approaches, the SRE mechanism produced signatures with more precise

and perfect matches to polymorphic worms. The previous approaches, whether autograph which generates a single contiguous byte string signature, or polygraph which generates token-based signature (bytes sequence), occur in significant numbers of suspicious flow pools. The autograph approach proves that the signature is not effective to match polymorphic worms. Polygraph proves that the signature depends on a token-based signature and it can lose vital information associated with any sequential token such as distances [14]

### 3.3 Needleman-Wunsch Algorithm

The Needleman-wunsch is one of the techniques to align protein or nucleotide sequences in bio-informatics is Needleman-Wunsch algorithm. It was one of the first applications of dynamic programming to compare biological sequences. The algorithm was developed by Saul B. Needleman and Christian D. Wunsch and published in 1970. The algorithm essentially divides a large problem into a series of smaller problems and uses the solutions to the smaller problems to reconstruct a solution to the larger problem. It is also sometimes referred to as the optimal matching algorithm and the global alignment technique. The Needleman-Wunsch algorithm consists of three steps :

- Initialisation of the score matrix.
- Calculation of scores and filling the traceback matrix .
- Deducing the alignment from the traceback matrix.

The Needleman-Wunsch algorithm is a global alignment algorithm that generates optimal alignment between a pair of sequences [27]. It aims to maximize the similarity score function given in Formula 3.1. Here  $k_m$  is number of matches,  $W_m$  is a score for character match.  $k_d$  is the number of mismatches.  $W_d$  is the score for mismatching characters.  $k_g$  denotes number of gaps.  $\delta$  is the penalty score for gaps.

$$SC(x, y) = k_m \times W_m + k_d \times W_d + k_g \times \delta \quad (3.1)$$

Sequence alignment is a process of comparing a pair or more of sequences. It searches a series of individual characters that are in same order in the sequences. Here a sequences is written on the top of another sequence with most positions having common characters.

Seq1 : PIDxxxRCVxxxxACKNLGxxxx

Seq2 : dsfPIDdsdRCVvvvACKNLGb

-	-	-	P	I	D	x	x	x	R	C	V	x	x	x	x	A	C	K	N	L	G	x	x	x	x
d	s	f	P	I	D	d	s	d	R	C	V	v	v	v	-	A	C	K	N	L	G	b	-	-	-
*	*	*	P	I	D	?	?	?	R	C	V	?	?	?	*	A	C	K	N	L	G	?	*	*	*

Figure 3.1: Sequence Alignment Result

According to above sequences and Figure 3.2 the similarity score of the given example is  $5(12 * 1 + 7 * 0 + 7 * 1)$ , where we set  $W_m = 1$   $W_d = 0$ ,  $\delta = -1$ . But if there is a lot of single length substrings, the Needleman-Wunsch algorithm results in a large number of pieces instead of the invariant part of the worm. So the Needleman-Wunsch algorithm needs to be modified to avoid such cases.

### 3.4 Smith-Waterman Algorithm

The Smith-Waterman algorithm is a dynamic programming method for determining similarity between nucleotide or protein sequences. The algorithm was first proposed in 1981 by Smith and Waterman and is identifying homologous regions between sequences by searching for optimal local alignments. To find the optimal local alignment, a scoring system including a set of specified gap penalties is used [26]. The Smith-Waterman algorithm is build on the idea of comparing segments of all possible lengths between two sequences to identify the best local alignment. This means that the Smith-Waterman search is very sensitive and ensures an optimal alignment of the sequences. The Smith-Waterman algorithm is the most accurate algorithm when it comes to search databases for sequence homology but it is also the most time consuming . it also consists of three steps :

- Initialization of a matrix.
- Matrix Filling with the appropriate scores.
- Trace back the sequences for a suitable alignment

### 3.5 Tree-Base Algorithm

Feng and Doolittle (1987) proposed a progressive alignment algorithm (tree-base algorithm), which uses the method of Needleman and Wunsch and for constructing an evolutionary tree(Bhattacharjee et al., 2006) to know the relationship between sequences [32]. The progressive alignment algorithms perform it operation through branching order of a guide tree and thus often get trapped to local optima (Nazninet al., 2012). To avoid such kind of local optima it is suggested in the literatures to use either stochastic or iterative procedure (Mohsen et al., 2007; Gotoh, 1982).

### 3.6 Yadav and Banka's Algorithm

By referring to previous studies (Devereux et al., 1984; Jagadamba et al., 2011; Nguyen and Yi, 2011; Katoh et al., 2005; Pei and Grishin, 2007; Li et al., 2004; Ma et al., 2002; Pearson, 2000), it can be concluded that none of the existing algorithms were accurate enough to provide an optimal alignment for all the datasets. As a result, with the uses of iterative refinement strategies (Gotoh, 1982), Genetic Algorithms (Peng et al., 2011) an iterative algorithms (Mohsen et al., 2007) were developed to construct more reliable and efficient multiple alignments which is used by Yadav and Banka in their paper. Also, all these methods listed above have shown their superiority in aligning distantly related sequences for a variety of datasets (Blackshields et al., 2006; Thompson et al., 1999). However, some accuracy was degraded while considering the distantly related sequences. The previous studies gives a clear indication that none of the method listed above can provide an accurate or meaningful alignment in all possible situations, irrespective of their advantages or disadvantages. Progressive alignment methods are known to be very fast and deterministic, but it suffers from a problem in which if any error occurs in the initial alignment and somehow gets propagated to other sequences than it cannot be corrected. However, this type of problem does not exist for iterative methods. In general, iterative methods are much slower in comparison to progressive methods and are used in a place where the best possible alignment is of prime importance and not the computational cost.

Evolutionary algorithms such as the genetic algorithm, which are based on the natural selection processes, are used for implementing iterative methods. Such algorithms have an upper edge with respect to others in the sense that these algorithms are independent for any types of scoring function. This gives an independence that without much alteration to the alignments, different objective functions can easily be tasted. Also, evolutionary algorithms can give low-cost clusters and multi-core processors because of they can be easily parallelize to meets the current trend. In this study, genetic algorithms (Pengfei et al., 2010) has been considered for experimental analysis. The main advantage of using GA for MSA problem is that it does not requires any particular source of algorithm to solve a given problem. Only, requirement for GA is the fitness function (Dongardive and Abraham, 2012), for necessary analysis and evaluation of solutions. Because GA is an highly implicitly parallel technique therefore, it can be used to solve various large scale and real time problems such as the travelling sales man problem (Zhang and Wong, 1997; Ulder et al., 1991). For a sequences of smaller length it can be possible to do the alignment manually but sequences of larger length requires an algorithm for successful alignment. Progressive alignment technique such as the dynamic programming (DP) suffers from a problem of early convergence or local optima problem and hence cannot be used for alignment of larger sequences [28]. Since they give priority to GA over DP . The main concept of genetic algorithm is show below :

The algorithm begins by generating the random population of n chromosomes.

i) Fitness function  $f(x)$  is evaluated for each chromosome x in the population.

ii) Repeat the following steps to create a new population

- Select parent based on the fitness value.
- Produce children by combining two parents using crossover operator and make changes in the resultant chromosome using mutation operator.
- Calculate the fitness value for each chromosome
- Generate new population by replacing the population with the children.

Initial generation, fitness function, crossover and mutation are four operators of genetic algorithm. In initial generation step initial solution is generated by putting gap between residues. In second step for measuring fitness of multiple sequence alignments the SPM is used. Crossover is done using single point crossover process. The main part which has been focused in their paper is mutation operator. They selected one individual randomly from whole population and then use two distance matrix dynamic and edit distance matrix for measuring distance between two sequences. Then randomly picked one distance matrix to construct guide tree using UPGMA method. They tested their proposed model on Bali base with some of the existing methods such as HMMT, ML-PIMA and found that their proposed method perform better in most of the cases [33].

### 3.7 CSR (Contiguous Substrings Rewarded) algorithm

In Tang et al. a pairwise sequence alignment is proposed named CSR(contiguous substrings rewarded). This algorithm consists of three parts:

- **Rewarding contiguous substrings**

Tang et al modified the similarity function from Formula 3.1 to Formula 3.2 including a score function  $enc()$  to reward contiguous substrings. They define  $enc(x) = 3(x-1)$  and set  $W_m = 0.5$ ,  $W_d = 0$ ,  $=-1$ . So far ,the similarity function results in 32  $(0.5*12+0*7+-1*7+3*(12-1))$ . Thus the CSR algorithm will score better for Figure 3.2.

$$SC(x, y) = k_m \times W_m + k_d \times W_d + k_g \times \delta + \sum enc(|s|) \quad (3.2)$$

- **Supporting wildcards**

The CSR algorithm has two predefined wildcards '?' and '\*'. '?' and '\*' character provided in the algorithm of Tang et al. by a Table 1 considering comparison rules and also '-' is considered as a gap in alignment. In Table 1 x and y are the characters in input sequences and r is alignment result sequence.

<i>x</i>	$\alpha$	$\alpha$	$\alpha$	?	$\alpha$	?	$\alpha$	—	*	$\alpha$	?
<i>y</i>	$\alpha$	$\beta$	?	?	?	*	*	*	*	—	—
<i>r</i>	$\alpha$	?	?	?	?	*	*	*	*	*	*

Figure 3.2: Sequence Alignment Result

- **Preserving distance restriction**

For preserving distance restriction of polymorphic worm signature Tang et al assign a length area [min, max] to each character during alignment where min is the lower bound and max is the upper bound. And finally the length range of a character is determined by minimizing lower bound and maximizing upper bound.

After detecting the SRE signature this algorithm is iteratively employing for the alignment for more than two sequences. If X, Y and Z are three sequences taken as input then this technique first aligned X and Y, then using the result, it aligns the third sequence.

### 3.8 M-STR-EC-LCS Algorithm

Yingjie Wu, Lei Wang, Daxin Zhu and Xiaodong Wang [23] provided solution for generalized longest common subsequence problem with multiple substring exclusive constraints. Previously it was supposed to be an NP-hard problem. But the authors of this paper succeeded to solve this in polynomial time complexity. This algorithm uses Dynamic Programming technique to solve the LCS problem. To exclude the constraints, a keyword tree is introduced. This keyword tree is used for string matching to determine the existence of constraint in the substring and leave it for LCS calculation.

The time complexity of the algorithm is  $O(nmr)$ , where n and m are lengths of two input sequences and r is the total length of constraints.

### 3.9 Beam Search Algorithm

This algorithm is developed by Christian Blum, Maria J. Blesa and Manuel Lpez-Ibez [24] for the longest common subsequence problem. Beam search is a heuristic search algo-



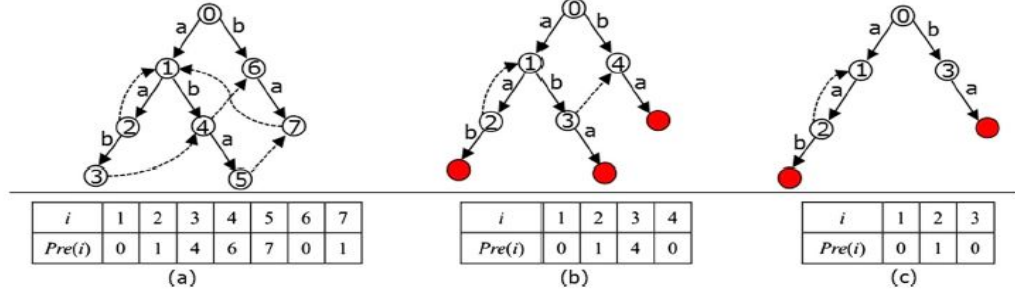


Figure 3.3: Keyword Trees

algorithm that explores a graph by expanding the most promising node in a limited set. Beam search is an optimization of best-first search that reduces its memory requirements. In beam search, only a predetermined number of best partial solutions are kept as candidates. Beam search uses breadth-first search to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. However, it only stores a predetermined number,  $k$ , of best states at each level (called the beam width).

Blum, Manuel and Blesa used this beam search approach to reduce the number of nodes traversed during LCS calculation. They developed two heuristic values to determine which nodes to keep and an upper bound to restrict the depth of the search tree. The heuristics are:

$$\eta_1(a) = \min\{|s_i| - p_i^a | i = 1, \dots, n\}, \forall a \in \sum_{nd} \quad (3.3)$$

$$UB(t) = |t| + \sum \min\{|y_i| | i = 1, \dots, n\} \quad (3.4)$$

The time complexity of this algorithm is  $O(\text{length}_{LCS} \times k_{bw})$ . Which gives a far better performance than the  $O(bd)$  time complexity of BFS search.

### 3.10 ECSR (Enhanced CSR) Algorithm

Aljawarneh et al. [5] modified the CSR algorithm provided by Tang et al. For sequence alignment purpose. The first step followed in this mechanism is to obtain the network flows from polymorphic worm and transform these into character sequences, which are considered as the worm samples. Aljawarneh et al. used Perl script to extract the optimal alignment score from the given sequences as well as worm sequences, which is expected to produce accurate outputs. For implementing SRE signature, it considered two steps, matrix fill step and trace back step. As using divide and conquer method, Needleman algorithm's formula is used to fill the alignment matrix and in trace back step, it captures the present last cornered



value and observes towards its predecessors from the neighbor cells.

For sequence alignment purpose, Aljawarneh et al. use iterative LCS (longest common substring) by divide and conquer technique. This algorithm finds the longest common substring between two sequences and then divides the sequences into two parts- left side and right side of the longest common substring. This Enhanced CSR algorithm emphasizes on the length of common sequence instead of similarity scoring. For example, let two sequences are seq1 = “oanaeazatwoa” and seq2 = “btwobobnbebz”. At first step, it finds the LCS “two” between seq1 and seq2. Then it looks for LCS in left side “oanaeaza” and “b” and in right side “a” and “bobnbebz”. None of them has any LCS. So at last, align the common substrings and add “-” in blank spaces.

o	a	n	a	e	a	z	a	t	w	o	-	-	-	-	-	-	-	a	
-	-	-	-	-	-	-	b	t	w	o	-	b	o	b	n	b	e	b	z
*	*	*	*	*	*	*	?	t	w	o	*	*	*	*	*	*	*	*	?

Figure 3.4: Enhanced CSR Result

They put “\*” in places of gaps and “?” in places of mismatch and include the distance restriction to generate the SRE signatures. So the worm signature is: -

“[1, 8]two[1,8]”

# CHAPTER 4

## PROPOSED METHOD

### 4.1 Polymorphic Worm Signature Detection

We propose an algorithm based on all common substrings between two strings to determine invariant parts of polymorphic worm shell code. These common substrings are then arranged in sequence to keep the schematics and characteristics intact. The proposed method also keeps the common substrings stored in a “Saved String” to prevent unwanted removal of invariant parts due to mismatched sequence, which enhances the accuracy of the proposed method.

#### 4.1.1 Proposed Algorithm

This algorithm takes two virus streams as input and shows the worm signature in SRE format as output. It also stores the common substrings found between the sequences so that when there is a new virus stream input, the algorithm can compare it with the stored substrings. The proposed algorithm follows:-

Step 1, Take the longest common substring from the two input streams using dynamic programming.

Step 2, Mark the positions of the substring in both the streams.

Step 3, Check the sequence of the substring with previously saved substrings (if any) in both the input streams.

Step 4, If the substring is in sequence with the previous substrings, store it in a data structure along with its starting position and finishing position of both the input streams.

Step 5, Find next longest common substring between the unmarked parts of the input strings.

Step 6, If there is any common substring, go to Step 2.

Step 7, Generate the distance restriction from the stored starting and finishing positions for each of the substrings to generate SRE signature.

Finally, If there is a new input stream, begin to compare the saved substrings with it using

the above steps. The resultant SRE signature indicates the invariants of polymorphic worm.

#### 4.1.2 Time & Space Complexity

The proposed method results in  $O(mnr)$  time complexity and space complexity where  $m$  and  $n$  denotes the length of the two input strings and  $r$  denotes the total length of the resultant common substrings.

#### 4.1.3 Simulation of Proposed Algorithm

Let us consider three sequences,

Seq1:        oxnxexzxtwox  
Seq2:        ytwoyoyneyyz  
Seq3:        oznzez

At first, we find a signature between Seq1 and Seq2. We find the longest common substring “two” from the both strings. We mark the positions of “two” in both sequences. As there are no more common substrings in saved strings data structure before so we dont need to check the sequence. We store “two” in saved strings. And find the next longest common substrings “o”, “n”, “e”, “z” respectively. As none of them are in sequence with “two” so we dont need to consider them as SRE signature. But they remain in the saved strings data structure. So from these two strings, the generated SRE signature is “[1,8]two[1,8]”.

Then when there is a new sequence as Seq3, we begin to compare the new sequence with our saved strings data structure according to our algorithm. This time common substrings are “o”, “n”, “e” and “z”. So the signature “two” was not actually an invariant part as it is not present in the Seq3. So the actual worm signature is “[0,3]o[1,1]n[1,1]e[1,2]z[0,0]”.

Another example is:-

Seq1:        bbbbobbbbxnxbbbbexbbbbbzxbbbtwoxbbbthreebbbb  
Seq2:        aaaytwoaaaayoyaaaanyeyzavvvavathreeaaaa  
Seq3:        ozjjjjjnzccjjjzkcckcckthreeckkkckckcc

At first, we find a signature between Seq1 and Seq2. We find the longest common substring “two” from the both strings. We mark the positions of “two” in both sequences. As there are no more common substrings in saved strings data structure before so we dont need to check the sequence. We store “two” in saved strings. And find the next longest common substrings “o”, “n”, “e”, “z” respectively. As none of them are in sequence with “two” so we don’t need to consider them as SRE signature. But they remain in the saved strings data structure. So from these two strings, the generated SRE signature is “[1,8]two[1,8]”.

Then when there is a new sequence as Seq3, we begin to compare the new sequence with our saved strings data structure according to our algorithm. This time common substrings are “o”, “n”, “e” and “z”. So the signature “two” was not actually an invariant part as it is not present in the Seq3. So the actual worm signature is “[0,12]o[1,6]n[1,6]e[1,4]z[5,7]three[4,10]”.

## 4.2 An Improved Genetic Algorithm for MSA Problem

There are various steps in Genetic Algorithm. Initial generation , generates new generation using genetic operators and the stopping criteria. But in our proposed method we will only work with crossover and mutation operators and the rest of the criteria are remained same as Yadav and Bankas method. These two steps are describe below.

### 4.2.1 Crossover

Two individuals chromosome of the parent A and the same chromosomes of the parent B are selected for crossover. This method is widely used to muddle the different characteristics of an individuals to create children out of parents obtaining the characteristics directly. In this process first we choose n points randomly in a chromosome. After that we can create the child. First considering the child A, until the first crossing point is found child A gets the first genes of parent A. Then rest genes of parent B are taken to the chromosome of child A until the next crossing point is found. This method is continued until the whole chromosome of child A is produced. In the similar way the child of B chromosome is created but starting taking the genes of parent B first [31] . The whole procedure is shown in figure 4.1 ,4.2,4.3,4.4 .



Figure 4.1: Parent chromosome

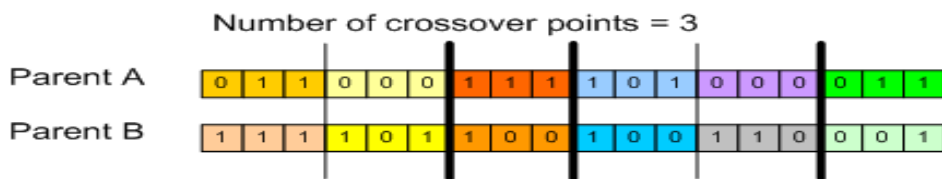


Figure 4.2: Multipoint crossover

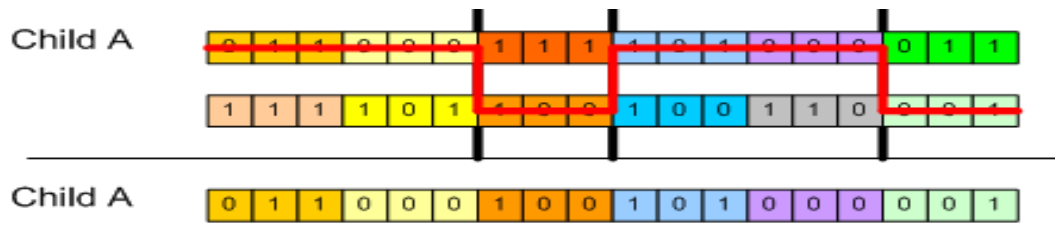


Figure 4.3: Child B chromosome

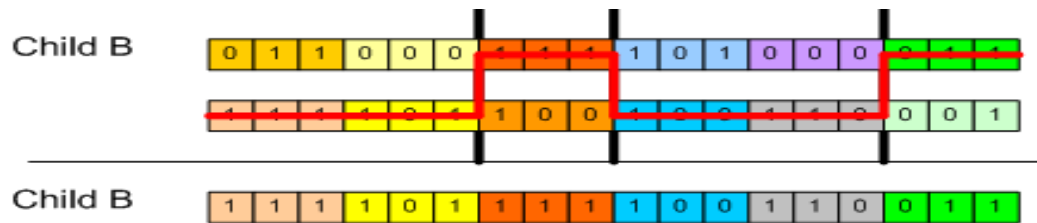


Figure 4.4: Child A chromosome

#### 4.2.2 Mutation

In Yadav and Bankas proposed model they have used edit distance matrix for building guide tree using UPGMA method. Their edit distance matrix had only used INDEL operations where in our proposed model we have used the swap operation in edit distance matrix which will minimize the total number of operation than Yadav and Bankas's model . We are taking an example of MSA problem to show how we calculate edit distance matrix using swapping operations.

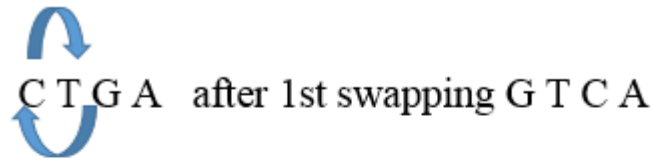
GTAC

CTGA

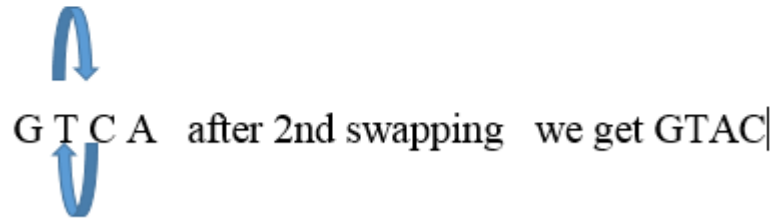
CGTA

Above the three strings considering the first pair of strings

GTAC CTGA



C T G A after 1st swapping G T C A

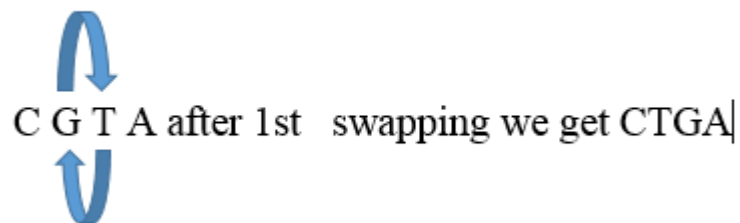


G T C A after 2nd swapping we get GTAC|

Figure 4.5: Converting CTGA string to GTAC

which is similar to first string. Two Minimum edit operations are needed for transform first sequence to second sequence whereas if we use INDEL operation we must have to perform 6 operations, three delete and three insert operations. So minimum edit distance between pair (1,2) is 2.

Now considering the second pair of strings CTGA CGTA



C G T A after 1st swapping we get CTGA|

Figure 4.6: Converting CGTA string to CTGA

which is similar to string CTGA . The same thing will require four INDEL operations which we are doing in one swapping operations. So minimum edit distance between pair (2,3) is 1.

Now comparing the first and third string, GTAC and CGTA

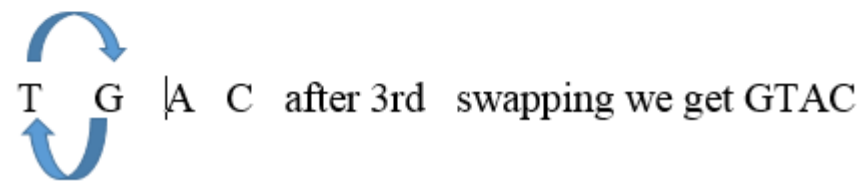
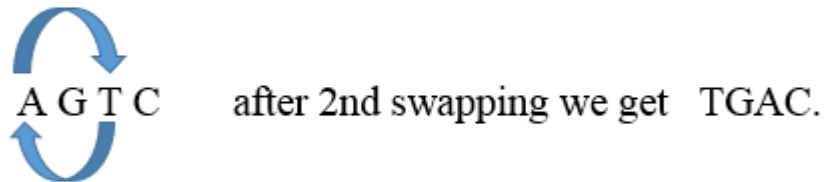
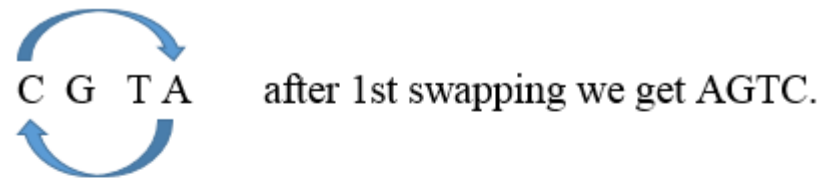


Figure 4.7: Converting CGTA string to GTAC

which is similar to our first string. Here we have to perform minimum three swapping operations whereas if we use INDEL we have to do eight edit operations, four delete and four insert operations. So minimum distance between pair (1,3) is 3. The whole distance between all pairs of string is shown in table I.

Table 4.1: Performance table of Edit distance matrix

	1	2	3
2	2	0	0
3	3	1	0

The minimum distance between all this pairs is distance between (2,3) that is 1. The combined sequences of all strings is shown in figure 4.8 .

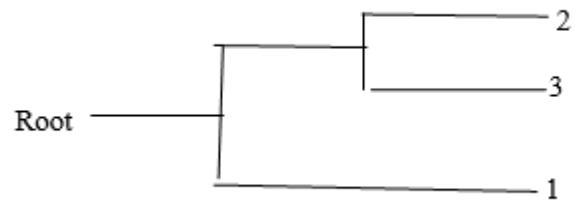


Figure 4.8: Combine sequence of all strings



# CHAPTER 5

## EXPERIMENTS AND RESULTS

### 5.1 Polymorphic Worm Signature Detection

#### 5.1.1 Introduction

In this section, the performance of our proposed method for polymorphic worm signature detection is analyzed. For experimental purpose, the proposed algorithm is implemented and demonstrated with random test cases. The results are then compared with two of the existing methods developed by Aljawarneh et al. and Tang et al. to show the improvement made by the proposed method.

#### Test Case Generator

As polymorphic worms are not open source and worm signatures have to be acquired through vendors, we created a synthetic polymorphic worm code generator. The characteristics of this generator is that it constantly maintains invariant parts like polymorphic worm and generates random data in-between the invariant parts. The invariant parts always keep their sequence unchanged but the random data does not. This test case generator was developed using C++ language.

#### Experimental Environment

For experimental purpose, the proposed algorithm was implemented in C++ language. We applied this implementation on the generated random test cases on a PC with 2.6GHz Intel core i5 processor having 4GB RAM running Windows 10 operating system to compare our results with Tang et al. and Aljawarneh et al.

#### Performance Parameters

We evaluated our algorithm based on two characteristics: Signature Accuracy and Similarity Scoring

- **Signature Accuracy:** We calculated the accuracy based on the equation 5.1 stated below:

$$Accuracy = \frac{T_c - U_c}{T_c} \times 100\% \quad (5.1)$$

Here,  $T_c$  = Total characters in the generated signature

$U_c$  = Unmatched characters compared with accurate signature

- **Scoring:** We used equation 5.2 for calculating the similarity score of the generated polymorphic worm signature.

$$SC(x, y) = k_m \times W_m + k_d \times W_d + k_g \times \delta + \sum enc(|s|) \quad (5.2)$$

### 5.1.2 Experimental Result

At first, we generated the worm signatures of the synthetic worm codes using the proposed algorithm, algorithm by Aljawarneh et al. and algorithm by Tang et al. The generated signatures are shown in Table 5.1.

Table 5.1: Generated Signatures

	INPUT STRINGS	Tang et al.	Aljawarneh et al.	Proposed Algorithm	
				OUTPUT	Saved Strings
1.	ifonehjad qponehjt qbhaconexvtwodgysfu	[2,2]one[1,1]j[3,5]	[2,2]one[1,1]j[3,5]	[2,2]one[1,1]j[3,5]	one, j
		[2,5]one[5,11]	[2,5]one[2,2]two[2,5]	[2,5]one[5,11]	one
2.	bkjmonelutwopsfrajc bsonervatwopqlyux cg100hvtipdfjmbki	[0,0]b[1,3]one[2,3]two[5,6]	[0,0]b[1,3]one[2,3]two[5,6]	[0,0]b[1,3]one[2,3]two[5,6]	one, l, u, two, b, s
		[2,10]two[0,4]p[5,6]	[2,8]two[0,5]p[5,7]	[2,10]two[0,5]p[5,7]	two, p, b, l
3.	onetwo twoone xxtwoxx	[0,3]one[0,3]	[0,3]one[0,3]	[0,3]one[0,3], [0,3]two[0,3]	one, two
		No string	[0,2]two[2,3]	[2,3]two[2,3]	two
4.	onepaclqmijqtwokuyvbrsxfh oytwoxlusi jkqirhflpaugonecdytwo	[0,0]o[1,9]two[1,2]u[0,4]s[1,5]	[0,0]o[1,9]two[1,2]u[0,4]s[1,5]	[0,0]o[1,9]two[1,2]u[0,4]s[1,5]	o, x, l, u, two, s
		[0,12]o[1,9]two[0,3]s[0,1]	[0,12]o[1,5]two[2,5]s[0,5]	[0,12]o[1,5]two[2,5]s[0,5]	o, two, s, l, u
5.	onetwox twoonex yytwo	[0,3]one[0,3]x[0,0]	[0,3]one[0,3]x[0,0]	[0,3]one[0,3]x[0,0]	one, two, x
		No string	[0,2]two[0,4]	[0,2]two[0,0]	two
6.	onextwox twoxoney onex	[0,4]twox[0,4]	[0,4]twox[0,4]	[0,4]twox[0,4]	twox, one
		No string	[0,4]one[0,1]	[3,3]one[0,1]	one
7.	oxnxexxtwox ytwoyoyneyez oznzez	[1,8]two[1,8]	[1,8]two[1,8]	[1,8]two[1,8]	two, o, n, e, z
		No string	[0,3]o[1,1]n[1,1]e[1,2]z[0,0]	[0,3]o[1,1]n[1,1]e[1,2]z[0,0]	o, n, e, z
8.	oneiagpxqdfgtwokcbjhlmv sciavgoneyfdxbtwtorju kmaquonexsvpyrjtowl	[0,5]one[3,6]d[2,2]two[1,3]j[1,4]	[0,5]one[3,6]d[2,2]two[1,3]j[1,4]	[0,5]one[3,6]d[2,2]two[1,3]j[1,4]	one, f, j, a, two, d, x
		[0,5]one[1,7]two[1,1]	[2,3]a[2,2]one[0,1]x[1,5]two[1,3]	[0,5]one[0,4]x[1,5]two[1,8]	one, j, a, two, x
9.	conekdjtwoipfng ltwomuptwoqrp jhdonefivqlksabcxmtwopyq	[1,7]two[1,2]p[2,5]	[1,7]two[1,2]p[2,5]	[1,7]two[1,2]p[2,5]	two, p
		[1,19]two[0,2]p[2,5]	[0,10]i[0,8]two[0,2]p[1,2]q[0,2]	[1,19]two[0,2]p[2,5]	two, p
10.	oneotwoyvax xonebvqrhaufyckigmasjltwop chjmonedalfwtobk	[0,1]one[1,18]two[1,4]	[0,1]one[1,18]two[1,4]	[0,1]one[1,18]two[1,4]	one, y, two, a
		[0,4]one[1,18]two[1,4]	[1,4]one[0,11]i[1,2]a[0,2]i[0,1]two[1,2]	[0,4]one[1,18]two[1,4]	one, y, two, a

We calculated the scoring of each randomly generated test cases according to equation 5.2 for our proposed algorithm and compared the scores with results from algorithm by Tang et al. and Aljawarneh et al. which is shown in Table 5.2 .

Table 5.2: Scoring Result Comparison

INPUT STRINGS		Tang et al.	Aljawarneh et al.	Proposed Algoritlm
1.	ifonehjadb	$4*.5+2*(-1)+3(4-1)=9$	$4*.5+2*(-1)+3(4-1)=9$	$4*.5+3(4-1)=11$
	qponehjtwwmk qbhaconexvtwodgysfu	$3*.5+9(-1)+3(3-1)=-1.5$	$7*.5+6(-1)+3(7-1)=15.5$	$3*.5+3(3-1)=7.5$
2.	bkjmonelutwopsfajc	$8*.5+4(-1)+3(8-1)=21$	$8*.5+4(-1)+3(8-1)=21$	$8*.5+3(8-1)=25$
	bsonervatwopqlyux cgtwohvqtlpdfjmbki	$3*.5+16(-1)+3(3-1)=-8.5$	$4*.5+12(-1)+3(4-1)=-1$	$4*.5+3(4-1)=11$
3.	onetwo	$3*.5+6(-1)+3(3-1)=1.5$	$3*.5+6(-1)+3(3-1)=1.5$	$3*.5+3(3-1)=7.5$
	twoone xxtwoxx	$3*.5+6(-1)+3(3-1)=1.5$	$.5*3+3(-1)+3(3-1)=4.5$	$3*.5+3(3-1)=7.5$
4.	onepaclmqijqtwokuyvbrsxfhdr	$6*.5+19(-1)+3(6-1)=-1$	$6*.5+19(-1)+3(6-1)=-1$	$6*.5+19(-1)+3(6-1)=-1$
	oytwoxlusi jkqirhflpaugonecdytwosm	$5*.5+33(-1)+3(5-1)=-19.5$	$6*.5+19(-1)+3(6-1)=-1$	$5*.5+3(5-1)=14.5$
5.	onetwox	$4*.5+6(-1)+3(4-1)=5$	$4*.5+6(-1)+3(4-1)=5$	$4*.5+3(4-1)=11$
	twoonex yytwo	$4*.5+6(-1)+3(4-1)=5$	$3*.5+6(-1)+3(3-1)=1.5$	$3*.5+3(3-1)=7.5$
6.	onextwox	$4*5+8(-1)+3(4-1)=3$	$4*5+8(-1)+3(4-1)=3$	$4*5+3(4-1)=11$
	twoxoney onex	$12(-1)=-12$	$3*.5+4(-1)+3(3-1)=3.5$	$3*5+3(3-1)=7.5$
7.	oxnxexxtwox	$3*.5+14(-1)+3(3-1)=-6.5$	$3*.5+14(-1)+3(3-1)=-6.5$	$3*.5+3(3-1)=7.5$
	ytwoyoyneyyz oznzez	$17(-1)=-17$	$3*.5+6(-1)+3(3-1)=1$	$3*.5+3(3-1)=7.5$
8.	oneiagpxqdfgtwokcbjlhmv	$8*.5+14(-1)+3(8-1)=11$	$8*.5+14(-1)+3(8-1)=11$	$8*.5+3(8-1)=25$
	sciavgoneyfdxbtworju kmaquonexsvpyrjt看	$6*.5+17(-1)+3(6-1)=1$	$8*.5+14(-1)+3(8-1)=19$	$7*.5+3(7-1)=21.5$
9.	conekdjtwoipfhg	$4*.5+9(-1)+3(4-1)=2$	$4*.5+9(-1)+3(4-1)=2$	$4*.5+3(4-1)=11$
	ltwomuptwoqrp jhdonefivqlksabcxmtwopyq	$4*.5+23(-1)+3(4-1)=-12$	$6*.5+21(-1)+3(6-1)=-3$	$4*.5+3(4-1)=11$
10.	oneotwoyvax	$6*.5+21(-1)+3(6-1)=-3$	$6*.5+21(-1)+3(6-1)=-3$	$6*.5+3(6-1)=18$
	xonebvqrhaufyckigmasjltwop chjmoneidaltwobk	$6*.5+24(-1)+3(6-1)=-6$	$9*.5+19(-1)+3(9-1)=8.5$	$6*.5+3(6-1)=18$

We also generated Accuracy percentage table Table 5.3 which compares the accuracy of the generated signatures of the 3 algorithms.

### 5.1.3 Performance Analysis

We produced results for 25 test cases and generated Scoring comparison chart 5.1 and Accuracy comparison chart 5.2.

From Accuracy chart we can see, our system gives 100% accurate result for all 25 test cases. Whereas the system proposed by Tang et al. [6] fails to produce any output for test case 3, 5 and 6. For test case 8 and 13, it produces partially accurate result. The algorithm by Aljawarneh et al. [5] succeeds to produce accurate result for only 12 out of 25 test cases. This occurs because we keep all the candidates of invariant code, whether longest or not in the saved string. From the scoring chart, our system generated better scores for 20 of the 25 test cases. From the 5 test cases where our system has lower score than Aljawarneh et al. . We can see that our system has 100% accuracy in all those cases, whereas the system by Aljawarneh et al. has low accuracy in all those five test cases.

Table 5.3: Accuracy Percentage Comparison

Test Case No.	Tang et Al.	Aljawarneh et al.	Proposed Algorithm
1	100%	50%	100%
2	100%	100%	100%
3	0%	100%	100%
4	100%	83%	100%
5	0%	100%	100%
6	0%	100%	100%
7	0%	100%	100%
8	83%	87%	100%
9	100%	67%	100%
10	100%	100%	100%

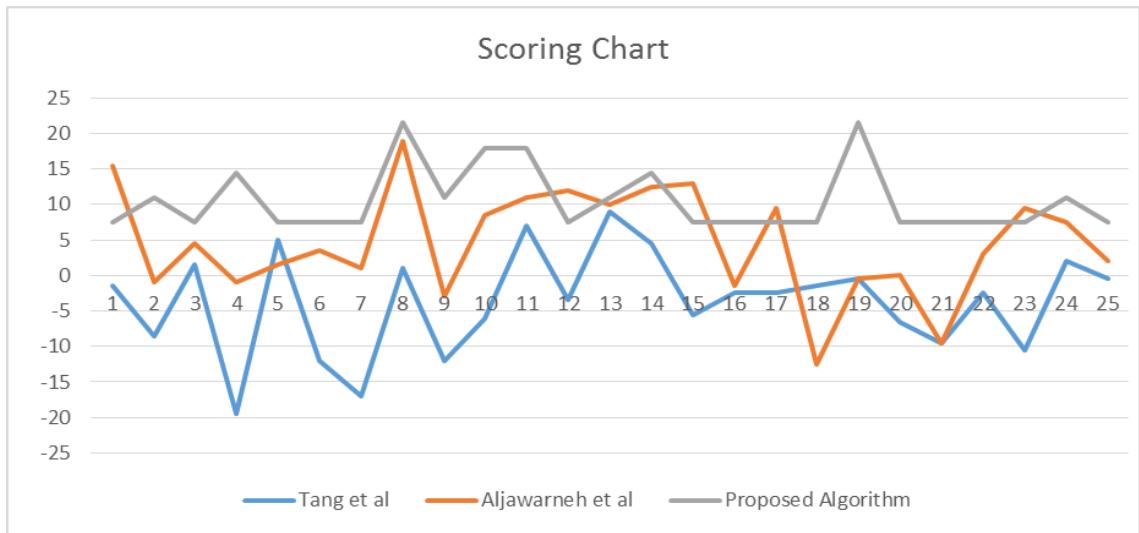


Figure 5.1: Scoring Comparison Chart

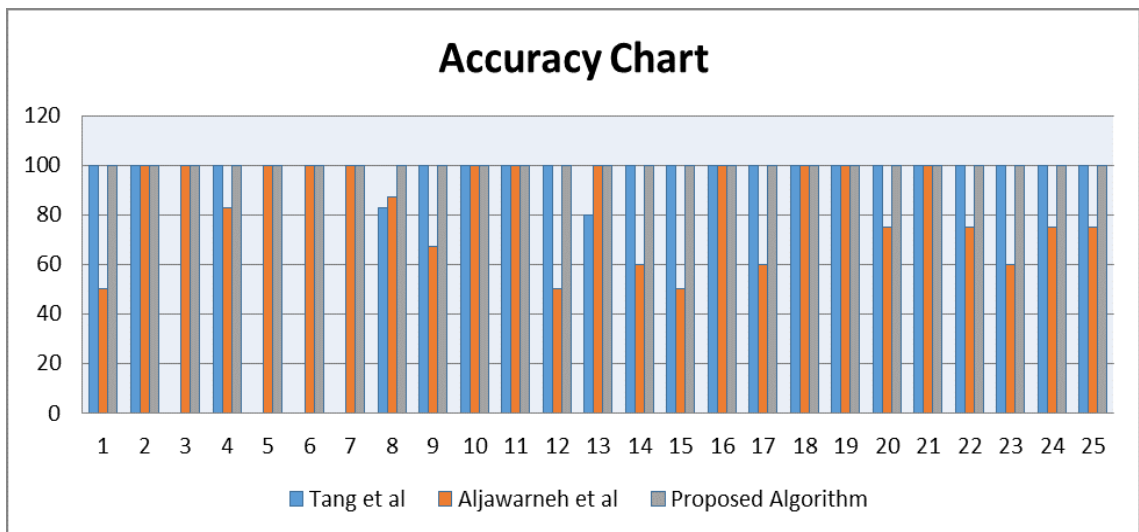


Figure 5.2: Accuracy Comparison Chart

The problem with our method is that it consumes a large amount of memory to store the common substrings for calculating accurate signature. Also space complexity is high due to dynamic programming tables for longest common substring.

# CHAPTER 6

## CONCLUSION

### 6.1 Conclusion

Our aim was to propose a new method of generating SRE based Polymorphic worm signature which will generate more accurate signatures with less false positive rate. So we introduce all common substring based signature generation algorithm which keeps distance restrictions and all essential features of polymorphic worm so that no invariant part is discarded coincidentally.

We also researched about mutation operator by considering improved edit distance matrix. In Yadav and Bankas proposed model they used edit distance matrix with only Indel operations and in our proposed model we have worked with swap operation which gives better or equal performance in most of the cases. A number of test cases and experiments have been shown to prove the efficiency of our proposed model. Based on the experiments we can say that our proposed model gives better or equal performance than Yadav and Bankas's model.

As we all know that the multiple sequence alignment is a known problem in bio-informatics, but still MSA remains a challenging task to explore. The arrangement of molecular sequences within an alignment to find similarities and differences among them is not an easy task, due to the complex size of the sequences and the search space. Because of the ability to handle complex scale problems, genetic algorithm is used as a genuine solution for the multiple sequence alignment problem. In this paper, a novel approach has been developed, which uses genetic algorithm with guide tree for performing multiple sequence alignment.

### 6.2 Future Work

We mainly focused on accuracy of polymorphic worm signature detection rather than minimizing the time complexity. In future we will try to determine the longest common substring using suffix tree instead of dynamic programming, which will reduce the time complexity from  $O(mnr)$  to  $O(mr+nr)$ . It will also result in reduction of space complexity as no DP tables are necessary. For our proposed MSA method, we plan to implement it and compare the results with existing methods to prove its efficiency.

## REFERENCES

- [1] Moftah, A. Maatuk, A. Plasmann, Aljawarneh, S.(2015). An Overview about the Polymorphic Worms Signatures. ICEMIS '15, September 24-26, 2015, Istanbul, Turkey 2015 ACM. ISBN 978-1-4503-3418-1/15/09.DOI: <http://dx.doi.org/10.1145/2832987.2833031>
- [2] Aljawarneh, S. 2011. A web engineering security methodology for e-learning systems. In Network Security, Elsevier. vol. 2011(3),pp. 12-15. doi:10.1016/S1353-4858(11)70026-5
- [3] Kruegel, C. Kirda, E. Mutz, D. Robertson, W. Vigna G. (2006). Polymorphic Worm Detection Using Structural Information of Executables. [Online]. Available from: <http://www.springerlink.com/content/f1k63052u27p6438/> [Access: 6 July 2010] .
- [4] Saudi, MM. Tamil, EM. Nor, SAM. Idris, MYI. Seman, K. (2008). EDOWA Worm Classification. In: Proceedings of the World Congress on Engineering 2008 Vol I. WCE 2008. ISBN: 978-988-98671-9-5.
- [5] Aljawarneh, Shadi A., Raja A. Moftah, and Abdelsalam M. Maatuk. "Investigations of automatic methods for detecting the polymorphic worms signatures." Future Generation Computer Systems 60 (2016): 67-77.
- [6] Tang, Yong, Xicheng Lu, and Bin Xiao. "Generating simplified regular expression signatures for polymorphic worms." International Conference on Autonomic and Trusted Computing. Springer Berlin Heidelberg, 2007.
- [7] Noh, H. Kim, J. Yeun, C.Y. Kim, K. (2008). New Polymorphic Worm Detection based on Instruction Distribution and Signature.
- [8] Bayoglu, B. Sogukpinar, I. (2008). Polymorphic Worm Detection Using TokenPair Signatures. [Online]. Available from: <http://www.pdfact.com/worm-detection-using-local-networks.pdf> [Access 27 August 2010]
- [9] Xiao, B. Tang, Y. Luo, J. Wei, G. (2009). Concept, characteristics and defending mechanism of worms. IEICE Transactions on Information and Systems; E92-D (5):799809.
- [10] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms", IEEE Security and Privacy Symposium, 2005.
- [11] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, B. Chavez, "Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience", In Proceedings of the IEEE Symposium on Security and Privacy, 2006

- [12] V. Yegneswaran, J. Giffin, P. Barford, and S. Jha, "An architecture for generating semantic-aware signatures", In USENIX Security Symposium, 2005.
- [13] Y. Tang and S. Chen, "Defending against internet worms: A signature-based approach", In Proceedings of Infocom, 2003.
- [14] Y. Tang and S. Chen, "An Automated Signature-Based Approach against Polymorphic Internet Worms", IEEE Transactions on Parallel and Distributed Systems, 2007
- [15] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, R.E Bryant, "Semantics-aware malware detection" , In IEEE Symposium on Security and Privacy, 2005.
- [16] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Polymorphic worm detection using structural information of executables", In Proceedings of Recent Advances in Intrusion Detection (RAID), 2005.
- [17] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: A basis for building self-protecting servers", In Proceedings of ACM CCS, 2005.
- [18] Kim, H.A. Karp, B. (2004). Autograph: Toward automated, distributed worm signature detection. In: USENIX Security Symposium; pp. 271286
- [19] Kim, H.A. (2010). Privacy-Preserving Distributed, Automated Signature-Based Detection of New Internet Worms. [Online]. Available from: <http://reports-archive.adm.cs.cmu.edu/anon/2010/CMU-CS-10-122.pdf> [Access 12 October 2010]
- [20] Newsome, J. Karp, B. Song, D. (2005). Polygraph: Automatically Generating Signatures for Polymorphic Worms. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy, pp. 226241. IEEE Computer Society Press, Washington
- [21] Tang, Y. Lu, X. Xiao, B. (2009). Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms. [Online]. Available from: <http://www.springerlink.com/content/f1k63052u27p6438/> [Access: 6 July 2010]
- [22] Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol Biol.48,44345(1970)
- [23] Wu Y, Wang L, Zhu D, Wang X. An efficient dynamic programming algorithm for the generalized lcs problem with multiple substring exclusive constraints. Journal of Discrete Algorithms. 2014 May 31;26:98-105.
- [24] Blum C, Blesa MJ, Lpez-Ibez M. Beam search for the longest common subsequence problem. Computers & Operations Research. 2009 Dec 31;36(12):3178-86.



- [25] “Aims of bioinformatics.” Last accessed on January 2, 2017, at 02.08.00 pm.[Online]. Available: <http://www.biology-today.com/bioinformatics-biostatistics/aims-of-bioinformatics>
- [26] “Bioinformatics explained: Smith-Waterman.” Last accessed on January 3, 2017, at 12.00pm. [Online]. Available: <http://www.montefiore.ulg.ac.be/~kvansteen/GBIO00091/ac20092010/Class4/Smith-Waterman.pdf>
- [27] Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48(3), 443453 (1970)
- [28] Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTALW: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*22(22), 46734680 (1994) .
- [29] Michalewicz, Z., Genetic algorithms + data structures = evolution programsThird, Revised and Extended Edition, 3 edn. Springer, Berlin (1996) .
- [30] Thompson, J.D., Plewniak, F., Poch, O.: A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res.*27(13), 26822690 (1999) .
- [31] “xatlantis@CyberTech” Last accessed on october 18, 2016 at 3pm. [Online]. Available: <http://www.xatlantis.ch/index.php/education/zeus-framework/49-genetic-algorithm>
- [32] Manish Kumar, AN ENHANCED ALGORITHM FOR MULTIPLE SEQUENCE ALIGNMENT OF PROTEIN SEQUENCES USING GENETIC ALGORITHM, EX-CLI Journal 2015 .
- [33] Rohit Kumar Yadav and Haider Banka, Genetic Algorithm Using Guide Tree in Mutation Operator for Solving Multiple Sequence Alignment, Springer India 2016.

# APPENDIX A

## ALGORITHMS

### A.1 Polymorphic Worm Signature Detection Algorithm

In Algorithm 1 we show how to find all common substrings between two strings and generate polymorphic worm signature. Algorithm 2 shows the procedure to generate signature from multiple strings.

---

**Algorithm 1** All Common Substring Search

---

```
1: procedure LCSubstring( $X[1...m], Y[1...n]$ )
2: struct{
3: int  $fslp, fsfp, lsfp, lslp$ ;
4: string  $name$ ;
5: } $allsub \leftarrow array()$ ;
6:  $index \leftarrow 1$ 
7:  $flag \leftarrow 0$ 
8:  $signature \leftarrow set\{\}$ 
9:  $mark\_X := array(1...m) \leftarrow \{0, 0, ..., 0\}$ 
10:  $mark\_Y \leftarrow array(1...n) \leftarrow \{0, 0, ..., 0\}$ 
11:  $L \leftarrow array(1...m, 1...n)$ 
12:  $result \leftarrow 0$ 
13: For  $i \leftarrow 1; i \leq m; i++$  do
14:     For  $j \leftarrow 1; j \leq n; j++$  do
15:     if  $i = 0$  Or  $j = 0$  then
16:          $L[i][j] \leftarrow 0$ 
17:     else
18:         if  $X[i - 1] = Y[j - 1]$  And  $mark\_X[i - 1] = 0$  And  $mark\_Y[j - 1] = 0$  then
19:              $L[i][j] \leftarrow L[i - 1][j - 1] + 1$ 
20:             if  $result < L[i][j]$  then
21:                  $result \leftarrow L[i][j]$ 
22:                  $a \leftarrow i - 1$ 
23:                  $b \leftarrow j - 1$ 
24:             end if
25:         else
26:              $L[i][j] \leftarrow 0$ 
27:         end if
28:     end if
29:     End For
30: End For
```

---

---

```

31:  $allsub[index].fsfp \leftarrow a - result + 1$ 
32:  $allsub[index].fslp \leftarrow a$ 
33:  $allsub[index].lsfp \leftarrow b - result + 1$ 
34:  $allsub[index].lslp \leftarrow b$ 
35: For  $i \leftarrow a - result + 1; i \geq a; i--$  do
36:    $mark\_X[i] \leftarrow 1$ 
37:    $allsub[index].name \leftarrow allsub[index].name + X[i]$ 
38: End For
39: For  $i \leftarrow b - result + 1; i \geq b; i--$  do
40:    $mark\_Y[i] \leftarrow 1$ 
41: End For
42: For  $i \leftarrow 1; i \leq index; i++$  do
43:   if  $allsub[index].fsfp > allsub[i].fsfp$  And  $allsub[index].lslp > allsub[i].lslp$  then
44:      $flag \leftarrow 1$ 
45:   else
46:     if  $allsub[index].fsfp < allsub[i].fsfp$  And  $allsub[index].lslp < allsub[i].lslp$ 
then
47:        $flag \leftarrow 1$ 
48:     else
49:        $flag \leftarrow 0$ 
50:        $return$ 
51:     end if
52:   end if
53: End For
54: if  $flag = 1$  then
55:    $signature \leftarrow signature \cup allsub[index]$ 
56: end if
57: end procedure

```

---



---

**Algorithm 2** Multiple String Common Substring

---

```

1: procedure M_STR_COM.SUB( $Z[1..p]$ )
2:   string  $sig$ 
3:   For  $i \leftarrow 1; i \leq index; i++$  do
4:      $sig \leftarrow sig + allsub[i].name$ 
5:   End For
6:    $LCSubstring(Z[1p], sig)$ 
7: end procedure

```

---

# APPENDIX B

## CODES

### B.1 Polymorphic worm detection code

```
1 #include<iostream>
2 #include<string.h>
3 #include<stdio.h>
4 using namespace std;
5 struct
6 {
7     string name;
8     int first_string_f_pos;
9     int first_string_l_pos;
10    int last_string_f_pos;
11    int last_string_l_pos;
12 } allsub[1000];
13 int allsub_index;
14 string result_str;
15 int X_in,Y_in;
16 int mark_X[1000],mark_Y[1000];
17 int max(int a, int b)
18 {
19     return (a > b)? a : b;
20 }
21
22 int LCSubStr(char *X, char *Y, int m, int n)
23 {
24     int LCSuff[m+1][n+1];
25     int result = 0;
26
27     for (int i=0; i<=m; i++)
28     {
29         for (int j=0; j<=n; j++)
```

```

30         {
31             if (i == 0 || j == 0)
32                 LCSuff[i][j] = 0;
33
34             else if (X[i-1] == Y[j-1])
35             {
36                 LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
37                 result = max(result, LCSuff[i][j]);
38             }
39             else LCSuff[i][j] = 0;
40         }
41     }
42     result=-100;
43
44     for (int i=0; i<=m; i++)
45     {
46         for (int j=0; j<=n; j++)
47         {
48             if (i == 0 || j == 0)
49                 LCSuff[i][j] = 0;
50
51             else if (X[i-1] == Y[j-1]&&mark_X[i-1]==0&&mark_Y[j-1]==0)
52             {
53                 LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
54                 if(result<LCSuff[i][j])
55                 {
56                     result=LCSuff[i][j];
57                     X_in=i-1;
58                     Y_in=j-1;
59                 }
60             }
61             else LCSuff[i][j] = 0;
62         }
63     }
64     allsub[allsub_index].first_string_f_pos=X_in-result+1;
65     allsub[allsub_index].first_string_l_pos=X_in;
66     allsub[allsub_index].last_string_f_pos=Y_in-result+1;
67     allsub[allsub_index].last_string_l_pos=Y_in;
68     for(int i=X_in-result+1; i<=X_in; i++)

```

```

69     {
70         mark_X[i]=1;
71         allsub[allsub_index].name+=X[i];
72     }
73     for(int i=Y_in-result+1; i<=Y_in; i++)
74     {
75         mark_Y[i]=1;
76     }
77     return result;
78 }
79 bool compare(int i)
80 {
81     if(
82         ((allsub[allsub_index].first_string_f_pos
83         <allsub[i].first_string_f_pos)&&
84         (allsub[allsub_index].last_string_f_pos
85         <allsub[i].last_string_f_pos))
86         ||
87         ((allsub[allsub_index].first_string_l_pos
88         >allsub[i].first_string_l_pos) &&
89         (allsub[allsub_index].first_string_l_pos
90         >allsub[i].last_string_l_pos))
91     )
92         return 1;
93     return 0;
94 }
95 int main()
96 {
97     char X[1000],Y[1000];
98     cin>>X>>Y;
99     cout<<X<<endl<<Y<<endl;
100     memset(mark_X,0,sizeof(mark_X));
101     memset(mark_Y,0,sizeof(mark_Y));
102     int m = strlen(X);
103     int n = strlen(Y);
104     int x=1;
105     allsub_index=1;
106     while(x>0)
107     {

```

```

108         x=LCSuSubStr(X, Y, m, n);
109         for(int i=1; i<allsub_index; i++)
110         {
111             if(!compare(i))
112             {
113                 allsub[allsub_index].name="";
114                 allsub_index--;
115             }
116         }
117         allsub_index++;
118     }
119     cout<<endl;
120     cout<<"Invarrient_part:_";
121     for(int i=1; i<=allsub_index; i++)
122     {
123         cout<<allsub[i].name<<"_";
124     }
125
126     return 0;
127 }

```

## B.2 Random test case generator code

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main()
4 {
5
6     freopen("in.txt", "r", stdin);
7     freopen("out.txt", "w", stdout);
8     char charset[30]="abcdefghijklmnopqrstuvwxyz"
9     ,modified_charset[30];
10    int length,number_of_fixed_string,col[200],
11    total_length=0,col_pos[1000],charset_cnt=0,emnei=0;
12    memset(col,0,sizeof(col));
13    memset(col_pos,0,sizeof(col_pos));
14    char fixed_string[1000][1000];
15    cin>>length;
16    cin>>number_of_fixed_string;

```

```

17     strcpy(modified_charset,charset);
18     for(int i=0; i<number_of_fixed_string; i++)
19     {
20         cin>>fixed_string[i];
21         total_length+=strlen(fixed_string[i]);
22
23         for(int j=0; j<strlen(fixed_string[i]); j++)
24         {
25             int l=0;
26             for(int k=0; k<26; k++)
27             {
28                 if(modified_charset[k]!=fixed_string[i][j])
29                     charset[l++]=modified_charset[k];
30                 strcpy(modified_charset,charset);
31             }
32             charset[l]='\0';
33         }
34
35     }
36
37     srand(time(NULL));
38     for(int i=0; i<100; i++)        ///nmber of string
39     {
40         cout<<i+1<<endl;;
41         for(int w=1; w<=3; w++)
42         {
43             char generated_string[1000];
44             memset(col_pos,0,sizeof(col_pos));
45
46             int temp_length=rand()%(length-10+1)+10;
47             ///length of generated string
48             int temp_nmber_of_fixedstring=number_of_fixed_string;
49             int t=temp_length;
50             for(int j=0;
51                 j<temp_length-total_length+number_of_fixed_string;)
52             {
53                 int pos=rand()%(temp_length+1);
54                 if(col_pos[pos]==0)
55                 {

```



```

56         if(temp_nmber_of_fixedstring>0&&
57            pos
58            +
59            strlen(fixed_string[temp_nmber_of_fixedstring-1])<t
60            )
61            {
62                for(int k=0,s=pos;
63                   k<strlen(fixed_string[temp_nmber_of_fixedstring-1]);
64                   k++,s++)
65                {
66                    generated_string[s]
67                    =
68                    fixed_string[temp_nmber_of_fixedstring-1][k];
69                    col_pos[s]=1;
70                }
71                temp_nmber_of_fixedstring--;
72                t=pos;
73                j++;
74            }
75            else
76            {
77                if(col[charset[emnei%20]]==0)
78                {
79                    generated_string[pos]=charset[emnei%20];
80                    emnei++;
81                    col_pos[pos]=1;
82                    j++;
83                }
84            }
85        }
86    }
87    generated_string[temp_length]='\0';
88    cout<<generated_string<<endl;
89 }
90 }
91 return 0;
92 }

```