Fall 2012

Simple Substitution Distance and Metamorphic Detection

Gayathri Shanmugam
San Jose State University

Simple Substitution Distance and Metamorphic Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Gayathri Shanmugam

December 2012

The Designated Project Committee Approves the Project Titled

Simple Substitution Distance and Metamorphic Detection

by

Gayathri Shanmugam

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2012

Dr. Mark Stamp	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science
Dr. Richard Low	Department of Mathematics

ABSTRACT

Simple Substitution Distance and Metamorphic Detection

by Gayathri Shanmugam

To evade signature-based detection, metamorphic viruses transform their code before infecting a new system. Software similarity measures are potentially useful as a means of detecting metamorphic malware. We can compare a given file to a known sample of malware and compute their similarity—if they are sufficiently similar, we classify the file as malware of the same family. The goal of this project is to analyze an opcode-based software similarity measure inspired by simple substitution cipher cryptanalysis.

ACKNOWLEDGMENTS

I would like to thank Dr. Mark Stamp, my project advisor, for his guidance, encouragement and support throughout the project.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Malicious software, or malware, comes in many different forms. For example, viruses and worms are types of malware that can replicate and spread from one computer to another without user permission or knowledge [25]. In this paper, we use the terms virus and malware interchangeably.

When the malware is executed, it might perform various malicious activities on a system, such as deleting files, overwriting important information, or stealing sensitive information such as passwords. Anti-virus (AV) developers apply a number of different techniques to detect malware [15]. The most widely used AV methodology is signature detection [17].

In the context of virus detection, a signature consisting of a sequence of bytes (possibly, including wildcards) is extracted from a known virus. Various string matching techniques are used to efficiently scan files for multiple signatures [1]. Since signature detection is the most commonly-used detection strategy, virus writers have developed many techniques designed to evade such detection. Arguably, metamorphic malware represents the most sophisticated technique developed to date for evading signature-based detection [2].

A metamorphic virus uses code transformations to morph its code at each infection. That is, the internal structure is altered, but the functionality remains unchanged. If the morphing is sufficient, then no common signature can be extracted from the viruses, and hence, well-designed metamorphic malware will evade detection by signature-based AV systems.

Previous research has shown that techniques based on software similarity are potentially useful as a means of detecting metamorphic malware. Similarity-based techniques classify a file as a virus if it is sufficiently similar to a member (or members) of the virus family [13]. The goal of this research is to analyze an opcode-based similarity measure inspired by simple substitution cipher cryptanalysis [6].

In a simple substitution cipher, each plaintext symbol is mapped to one ciphertext symbol. A fast hill climb attack on simple substitutions is given in [6]. This attack measures distance based on plaintext language digraph frequencies. Each putative plaintext is scored by computing the similarity of its digraph distribution to the expected digraph distribution of the plaintext. The algorithm is extremely efficient, since each modification of the key is mapped directly to a digraph distribution matrix, without any need to re-compute the putative plaintext. At each step, a score is obtained which measures how well the putative plaintext fits the plaintext language digraph statistics. We have applied an analogous technique to metamorphic viruses, based on extracted opcode sequences. For our approach, the score can be viewed as a measure of distance between a given file and a specific metamorphic family. Files that have scores indicating they are "close" to a metamorphic family will be classified as family viruses.

This paper is organized as follows. Chapter 2 gives background information on different types of malware and their detection techniques. In Chapter 3, we briefly discuss simple substitution ciphers, while Chapter 4 provides an overview of the fast attack on simple substitution ciphers [6] that forms the basis of the software similarity technique considered here. Chapter 5 discusses the proposed opcode-based similarity technique and Chapter 6 presents our experimental results. Chapter 7 provides our conclusion and a discussion of possible future work.

CHAPTER 2

Malware

Malware is software that is designed to break security [17]. Anti-virus (AV) software aims to detect and remove malware. Since the first AV software developed, there has been a constant battle between virus writers and AV developers [15]. As the AV softwares are advancing their methods, the virus writers are developing various techniques to produce malware that are more and more difficult to detect. The most commonly used techniques are encryption, polymorphism and metamorphism [16].

2.1 Encrypted Virus

An encrypted virus encrypts its body using different keys each time it propagates. Different keys will yield different virus copies. So the AV software cannot scan for the common signature in the virus copies [19]. The downside of this approach is that the encrypted virus must include the decryption code, which will remain constant across generations. Hence this code is subject to signature detection [17].

2.2 Polymorphic Virus

A polymorphic virus is an encrypted virus whose decryption code is morphed. As the decryption code varies from generation to generation, there will be no common signature. Hence it would not be recognized by the signature-based scanners [10].

However, polymorphic viruses can be detected using code emulators [15]. The code emulator executes the suspicious code on a virtual machine and examines its behavior. If the program is a malware, it will eventually decrypt itself in the virtual memory. Once decrypted, it can be detected using signature-based scanners, as all the

viruses carry the same virus body [14]. But this type of detection is slower than the standard signature detection due to the emulation process [17]. Figure **??** illustrates different generations of a polymorphic virus [21].

2.3 Metamorphic Virus

A metamorphic virus morphs its body before infecting a new system. The mutated virus will have the same functionally as the original worm, but they are structurally different. Metamorphic virus uses various advanced obfuscation techniques such as register swapping, subroutine permutation, garbage instruction insertion, instruction substitution, transposition etc. to produce morphed copies [24].

2.3.1 Code Obfuscation Techniques

The most commonly used obfuscation techniques employed by metamorphic viruses are presented in this section.

2.3.1.1 Register Swapping

Register Swapping is the simplest obfuscation technique employed by metamorphic viruses such as Win95/Regswap virus [20]. The virus mutates its body by using different registers, but the opcodes remain the same across generations. Figure 1 shows the code fragments from two generations of Win95/Regswap which is given in [21].

```
5A              pop  edx
BF04000000      mov  edi,0004h
8BF5            mov  esi,ebp
B80C000000      mov  eax,000Ch
81C288000000    add  edx,0088h
8B1A            mov  ebx,[edx]
899C8618110000mov  [esi+eax*4+00001118],ebx

58              pop  eax
BB04000000      mov  ebx,0004h
8BD5            mov  edx,ebp
BF0C000000      mov  edi,000Ch
81C088000000    add  eax,0088h
8B30            mov  esi,[eax]
89B4BA18110000mov  [edx+edi*4+00001118],esi
```

Figure 1: Different Generations of Win95/Regswap

2.3.1.2 Subroutine Permutation

Metamorphic viruses such as Win32/Ghost uses this technique to create morphed copies by reordering the subroutines. If there are n number of subroutines, then $n!$ different copies can be produced. Figure 2 shows the subroutine permutation with eight subroutines which is given in [21].

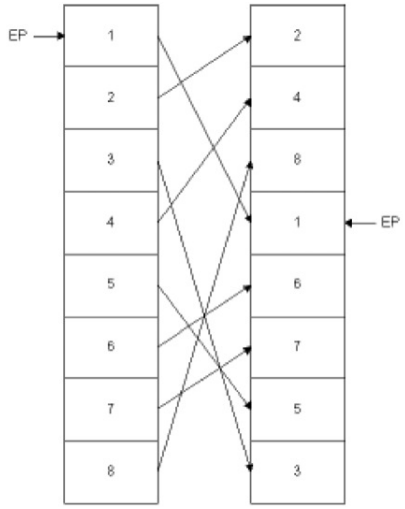

Figure 2: Subroutine Permutation with Eight Subroutines

2.3.1.3 Garbage Instruction Insertion

Metamorphic viruses such as Win95/ZPerm [21] uses insertion and removal of garbage instructions to create morphed copies. Garbage instructions also known as "do nothing code" are instructions that does not alter the functionality of the program when executed. eg: `NOP`. Using garbage instructions, a virus can produce an unlimited number of morphed copies [14, 16, 24].

2.3.1.4 Instruction Substitution

Certain metamorphic viruses generate morphed copies by replacing an instruction with another equivalent instruction. For example, `SUB EAX,EAX` will be replaced by `XOR EAX,EAX` in the morphed copy. Both the instructions mentioned above, have the same functionality but the opcodes are different.

2.3.1.5 Transposition

Some metamorphic viruses create morphed copies by changing the order of execution of instructions having no dependency between them. The morphed copies will have the same functionality, as the instructions that are re-ordered have no dependency between them. For example, the set of instructions

```
MOV R1,R2
ADD R3,R4
```

will be morphed as:

```
ADD R3,R4
MOV R1,R2
```

2.4 Malware Detection Techniques

As viruses evolve, there has been a corresponding advancement in virus detection techniques as well. This section gives an overview of some of the popular virus detection techniques.

2.4.1 Signature Detection

Signature detection is the most widely used AV technique [1]. An unique string of bits called "signatures" are extracted from all known viruses and is stored in the AV scanner database. Signatures are selected in such a way that it does not appear in any other softwares [17, 19]. The AV scanner scans the files to find the signature of the viruses. If a known signature is present in any of the files, then the file will be detected as the virus file. The downside of this method is that, it can only detect known viruses [15].

2.4.2 Code Emulation

Code Emulation is one of the strongest virus detection techniques [15]. It executes the suspicious program on a virtual machine and examines its behavior. Since the detector deals with the malicious code in a controlled environment, the risk of propagation is minimized [15]. Polymorphic viruses can be detected using this technique.

2.4.3 Anomaly Detection

Anomaly detection uses heuristics to detect unknown virus and variants of a known virus by analyzing the program's structure and its behavior instead of looking for signatures. Heuristic scanners can be classified as static or dynamic [15]. The

primary difference between the two types is that the dynamic scanner uses emulation to analyze the program behavior while the static one does the analysis on the actual system, but the operation of both the scanners is the same. The scanner performs the operation in two phases - analysis phase and detection phase [19]. During the analysis phase, the scanner scans the virus body and gathers all possible behaviors from the program under investigation. During the detection phase, it analyzes the observed behaviors and classify the suspected program as virus or not [11]. The downside of this approach is its high false alarm rate [5].

CHAPTER 3

Substitution Ciphers

Substitution ciphers are one of the oldest cipher systems [17]. In such a cipher, each plaintext symbol is substituted by a ciphertext symbol. The symbols include letters, digrams, trigrams etc. There are many different types of substitution ciphers. This section gives a brief overview of only one type, simple substitution ciphers.

3.1 Simple Substitution Ciphers

As its name suggests, simple substitution ciphers are the simplest of the substitution ciphers [17]. In this cipher, each plaintext symbol maps to one ciphertext symbol [8, 18].

A simple substitution key is given in Table 1. In this table, each ciphertext letter is obtained by shifting the plaintext letter 3 positions forward in the alphabet [17]. Hence the plaintext message `LETTER` encrypts to `OHWWHU`, if the key in Table 1 is used for encryption [2].

Table 1: Simple Substitution Key

plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ciphertext	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

If the plaintext is English, then the simple substitution cipher consists of $26! \approx 2^{88}$ possible keys. On an average, an attacker has to try 2^{87} to break the cipher using the brute force approach. Suppose the attacker can test 2^{40} keys per second, then the above key can be exhausted in $2^{87}/2^{40} = 2^{47}$ seconds, or about 4.4 million years. This implies that if the keyspace is big, then the brute force approach is highly impractical

on a simple substitution cipher. But a cryptanalyst uses English monograph statistics to break the cipher than using an exhaustive search. That is, the attacker's reasonable guess would be the most frequent ciphertext letter maps to the most frequent letter in English, which is `E`, the second most frequent ciphertext letter maps to `T` and so on. By proceeding in this way, the attacker will be able to recover most of the plaintext message. Then he/she can easily make guesses on the remaining ones [18].

CHAPTER 4

Fast Attack on Simple Substitution

An efficient algorithm to break simple substitution ciphers is given in [6]. It starts by making an initial guess to the key and the key is improved through a number of iterations. At each step, using a scoring function, it determines how close the current key is to the actual key. If the current key is better, then the algorithm retains it; otherwise it does not. This algorithm is a hill climb attack because it eliminate the keys that does not improve the score [6].

The algorithm in [6] uses only the digraph distributions of English and the ciphertext to solve the cipher. It is extremely efficient because the ciphertext is parsed only once to construct an initial digraph distribution matrix. In the subsequent iterations, key is modified by simply altering this matrix and not by decrypting the ciphertext using the new key and re-constructing the digraph distribution matrix [6].

4.1 Hill Climbing Technique

Hill Climbing is a mathematical optimization technique that starts with a solution to a problem and it tries to discover a better solution by slightly altering the putative solution [26]. The new solution is then scored to detect whether it is better than the previous solution or not. If it is better, then an incremental change is made to the new solution; otherwise no change is made. The above process is repeated until the key can no longer be improved [18, 26].

4.2 Overview of the Algorithm

Jackobsen's algorithm [6] assumes that the plaintext is in English and the ciphertext symbols include 26 English alphabets. It starts by guessing an initial key that maps the most frequent ciphertext letter to the most frequent letter in English, which is `E`, second most frequent ciphertext letter to `T` and so on.

In the iterated loop, the algorithm slightly modifies the current key and uses that key to decrypt the ciphertext. It then checks if the putative plaintext is closer to the expected English language than before [6]. If so, then the new key is retained for the next iteration; otherwise the old key is modified in a different way. The above process is repeated for $\binom{26}{2}$ iterations to make sure all pairs of elements in the key are swapped once.

The modification of putative key K at each iteration is explained in [18], which is shown below. Suppose, the putative key is $K = k_1, k_2, ..., k_{26}$, where K is a permutation of English alphabets. In the first iteration, all adjacent elements are swapped. i.e. k_1 is swapped with k_2, and so on. In the second iteration, the elements at distance two are swapped. i.e. k_1 is swapped with k_3 and so on. In the n^{th} iteration, the elements at distance n are swapped. The above procedure is illustrated diagrammatically in [18] which is shown in (1). In (1), '|' denotes the swap.

$$
\begin{array}{lccccccccc}
\text{iteration 1:} & k_1 \mid k_2 & k_2 \mid k_3 & k_3 \mid k_4 & \ldots & k_{23} \mid k_{24} & k_{24} \mid k_{25} & k_{25} \mid k_{26} & & \\
\text{iteration 2:} & k_1 \mid k_3 & k_2 \mid k_4 & k_3 \mid k_5 & \ldots & k_{23} \mid k_{25} & k_{24} \mid k_{26} & & & \\
\text{iteration 3:} & k_1 \mid k_4 & k_2 \mid k_5 & k_3 \mid k_6 & \ldots & k_{23} \mid k_{26} & & & & \\
\vdots & & \vdots & & & \iddots & & & & \\
\text{iteration 23:} & k_1 \mid k_{24} & k_2 \mid k_{25} & k_3 \mid k_{26} & & & & & & \\
\text{iteration 24:} & k_1 \mid k_{25} & k_2 \mid k_{26} & & & & & & & \\
\text{iteration 25:} & k_1 \mid k_{26} & & & & & & & &
\end{array} \tag{1}
$$

To alter the current key, the algorithm simplify modifies the digraph distribution matrix of the putative plaintext. The procedure is explained in detail at the end of this section.

The algorithm uses the following scoring function to determine how close the putative plaintext digraph distribution matrix is to the expected English language digraph distribution matrix. Suppose D = $\{d_{ij}\}$ represents the putative plaintext digraph distribution matrix and E = $\{e_{ij}\}$ represents the expected English language digraph distribution matrix, then the similarity of the matrices is given by the equation in (2). The similarity score is always greater than or equal to zero, with equality obtained for a perfect match.

$$score(K) = d(D, E) = \sum_{i,j} |d_{ij} - e_{ij}| \tag{2}$$

Pseudocode of the algorithm is given in [18] which is shown in Table 2.

Table 2: Fast Simple Substitution Attack [6, 18]

Algorithm parameters:
E matrix - expected English language digraph distribution matrix
$K = k_1, k_2, ..., k_n$ - initial putative key in the descending order of expected frequency
C - ciphertext
P - putative plaintext recovered from ciphertext C using key K
D matrix - digraph frequency matrix for P
score $d(D, E) = \sum_{i,j} \lvert d_{ij} - e_{ij} \rvert$

```
score = d(D, E)
for i = 1 to n − 1
    for j = 1 to n − i
        D′ = D
        swap rows j and j + i of D′
        swap columns j and j + i of D′
        if d(D′, E) < score then
            D = D′    //retain the swap
            swap(k_j, k_{j+i})    //swap the elements in the key
            score = d(D′, E)    //update the least score
        end if
    next j
next i
return K
```

The procedure to modify the key K is illustrated in [18], which is shown below. Let us take a simple substitution cipher which is based on only 10 English letters. The plaintext symbols in the descending order of their frequencies are

E,T,A,O,I,N,S,R,H and D.

Suppose the ciphertext is

TNDEODRHISOADDRTEDOAHENSINEOARDTTDTINDDRNEDNTTTDDISRETEEEEEAA. (3)

The monograph statistics corresponding to the ciphertext in (3) is

E	T	A	O	I	N	S	R	H	D
11	9	5	4	4	6	3	5	2	12

The initial putative key K is given in (4). The corresponding putative plaintext and the digraph distribution matrix is given in (5) and (6) respectively.

$$\begin{array}{rl} \text{Plaintext:} & \texttt{E T A O I N S R H D} \\ \text{Ciphertext:} & \texttt{D E T N A R I O S H} \end{array} \tag{4}$$

$$\texttt{AOETRENDSHRIEENATERIDTOHSOTRINEAAEASOEENOTEOAAAEESHNTATTTTTII.} \tag{5}$$

	E	T	A	O	I	N	S	R	H	D
E	3	1	2	1	0	3	1	1	0	0
T	2	4	1	1	1	0	0	2	0	0
A	2	2	2	1	0	0	1	0	0	0
O	2	2	1	0	0	0	0	0	1	0
I	1	0	0	0	1	1	0	0	0	1
N	1	1	1	1	0	0	0	0	0	1
S	0	0	0	2	0	0	0	0	2	0
R	1	0	0	0	3	0	0	0	0	0
H	0	0	0	0	0	1	1	1	0	0
D	0	1	0	0	0	0	1	0	0	0

(6)

The next step is to modify the putative key K. As discussed in the swapping procedure in (1), we first swap the first two elements. This gives us a new putative key, which is shown in (7). The corresponding putative plaintext and the digraph distribution matrix is given in (8) and (9) respectively.

$$\begin{array}{rl} \text{Plaintext:} & \texttt{E T A O I N S R H D} \\ \text{Ciphertext:} & \texttt{E D T N A R I O S H} \end{array} \tag{7}$$

$$\texttt{AOTERTNDSHRITTNAETRIDEOHSOERINTAATASOTTNOETOAAATTSHNEAEEEEEII.} \tag{8}$$

	E	T	A	O	I	N	S	R	H	D
E	4	2	1	1	1	0	0	2	0	0
T	1	3	2	1	0	3	1	1	0	0
A	2	2	2	1	0	0	1	0	0	0
O	2	2	1	0	0	0	0	0	1	0
I	0	1	0	0	1	1	0	0	0	1
N	1	1	1	1	0	0	0	0	0	1
S	0	0	0	2	0	0	0	0	2	0
R	0	1	0	0	3	0	0	0	0	0
H	0	0	0	0	0	1	1	1	0	0
D	1	0	0	0	0	0	1	0	0	0

(9)

It is clearly evident from the matrices (6) and (9) that, swapping the elements in the key is done by simply swapping the corresponding rows and columns of the putative plaintext digraph distribution matrix (D matrix), that begin or end with the swapped elements.

CHAPTER 5

Simple Substitution Distance for Metamorphic Malware Detection

For metamorphic malware detection, we intend to develop a hill climbing technique analogous to Jackobsen's algorithm [6], on extracted opcode sequences. The recovered "key" here can be viewed as a measure of the distance between the two opcode sequences. The basic idea is that we train the detection system on a sequence of opcodes extracted from a specific metamorphic family and the trained system will be used to score an unknown file to determine whether it belongs to the same virus family or not. In the remainder of this section, we discuss the design of this technique in detail.

Given an unknown executable file, we extract the opcode sequences from the code/text section of the file. We also extract the sequence of opcodes from the code section of a specific metamorphic family of viruses. We then construct two digraph distribution matrices, one using opcodes present in the unknown file and the other using opcodes present in the family viruses. Suppose n represent the number of distinct most frequently used opcodes. We map the opcodes to indices 0,1,2,..,n-1. Any opcode other than the top n that occurs in the family viruses or the benign files are grouped together under the same opcode category "Unknown". Since we consider only the top n, there will be very less number of zero entries in the matrices. That is, using this approach, the digraph matrices will not be sparse. Let $D = \{d_{ij}\}$ and $E = \{e_{ij}\}$ be the digraph distribution matrices of size $(n+1 \times n+1)$ of the unknown file and the family viruses respectively. Also, $\{d_{ij}\}$ and $\{e_{ij}\}$ represent the probability that opcode i is followed by opcode j in the unknown file and in the family viruses respectively. We experimented with different values of n and 25 proved to be the best

value for this technique. Both the matrices values are initialized to zero.

We choose an initial key K, that best matches with the monograph statistics of opcodes in the family viruses. That is, we assume the most frequent opcode in the family viruses maps to the most frequent opcode in the unknown file, second most frequent opcode in the family viruses maps to the second most frequent opcode in the unknown file and so on. We fill in the matrix D based on this initial key K. We then normalize the D matrix by dividing the count in each cell by the sum of all the cells in the matrix.

We construct E matrix using the following procedure. Suppose m denotes the number of distinct virus files under the metamorphic family, we construct m matrices of size $(n+1 \times n+1)$. We fill matrix 0 with the digraph frequency counts of opcodes in file 0, matrix 1 with the digraph frequency counts of opcodes in file 1 and so on. We then normalize the matrices, matrix 0 to matrix $m-1$ by dividing each cell in the matrix by the sum of all the cells in the matrix.
We then construct E matrix as follows.

$$E = \{e_{ij}\} = (matrix0_{ij} + matrix1_{ij} + ... + matrix(m-1)_{ij})/m \tag{10}$$

We experimented with another normalization technique to normalize the E matrix, but the above mentioned technique proved to give better results for this technique. To compare the matrices D and E, we compute the score using the scoring function in (11). Other scoring functions were tested, but the above mentioned one proved to give better results for this technique.

$$score(K) = d(D, E) = \sum_{i,j} |d_{ij} - e_{ij}| \tag{11}$$

In the iterated loop, we alter the putative key by swapping opcodes in the key $K = opcode_0, opcode_1, ..., opcode_{n-1}, opcode_{Unknown}$. Swapping procedure, which is

the same as in Jackobsen's approach [6], is as follows. In the first iteration, all the adjacent opcodes are swapped. That is, $opcode_0$ is swapped with $opcode_1$ and so on. In the second iteration, the opcodes at distance two are swapped, that is $opcode_0$ is swapped with $opcode_2$ and so on. In the n^{th} iteration, the opcodes at distance n are swapped. After each swap, we update the D matrix and compute the score by comparing the updated matrix with matrix E. If the score improves, we update the putative key and start over again from the first iteration. If the score does not improve, then we take the old key and try a different modification. We continue the swapping procedure for $\binom{n}{2}$ iterations to make sure all $\binom{n}{2}$ pairs of opcodes in the key are swapped once. If the resulting score is below the threshold, we classify the unknown file as virus of the same family. We experimented with different swapping strategies, but the above mentioned strategy proved to give better results for this technique.

Figure (3) and Table (3) show the flow diagram and pseudocode of the detection technique respectively.

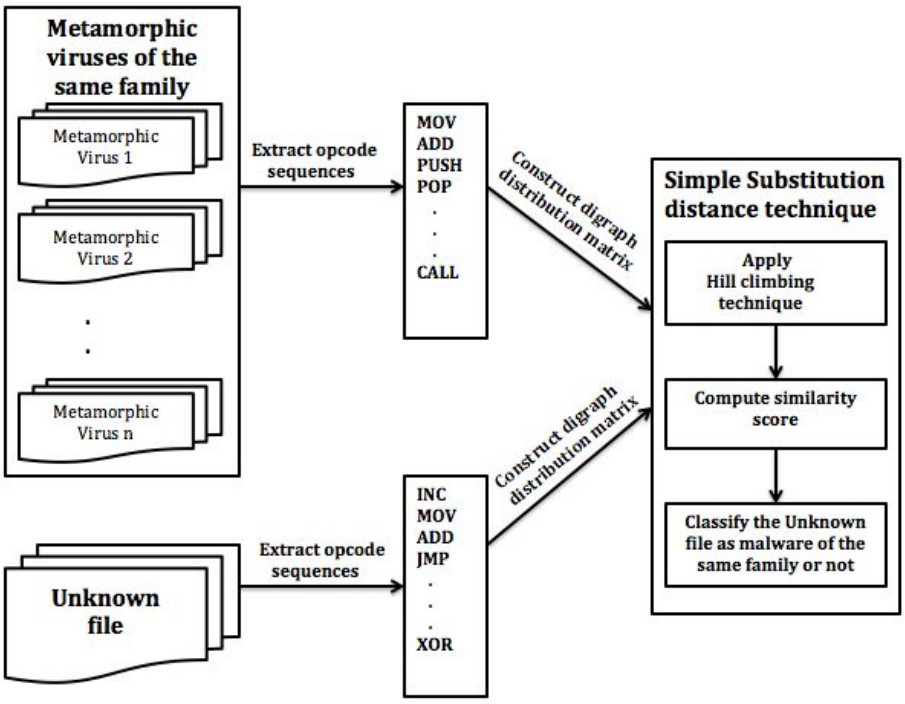

Figure 3: Flow Diagram of the Proposed Technique

Table 3: Fast Algorithm for Metamorphic Malware Detection

Algorithm parameters:
Extract opcode sequences from the unknown file and family viruses Determine top n distinct most frequently used opcodes E - digraph distribution matrix of a specific metamorphic family $K = opcode_0, opcode_1, ..., opcode_{n-1}, opcode_{Unknown}$ - initial putative key in the descending order of frequency D - digraph frequency matrix corresponding to an unknown file score - $d(D,E) = \sum_{i,j} \lvert d_{ij} - e_{ij} \rvert$

```
score = d(D, E)
for k = 1 to n
    for i = 0 to n − 1 and j = i + k to n − 1
        swap rows i and j of D
        swap columns i and j of D
        if d(D, E) < score then
            swap(opcode_i, opcode_j)      //swap the elements in the key
            score = d(D, E)      //update the least score
            k = 0 //start over from the first iteration
            break //start over from the first iteration
        end if
        else
            swap rows i and j of D //revert back to the old key
            swap columns i and j of D //revert back to the old key
        end else
    next i (i++) and next j (j+ = k)
next k
if score ≤ threshold
    unknown file is a virus of the same family
else
    unknown file is not a virus of the same family
```

To demonstrate the procedure to update the D matrix, we take a simplified example. Let us consider the matrices are of size (5×5) and the top 5 distinct opcodes that can be present in any kind of virus or benign files in the decreasing order of frequencies are

MOV	CALL	ADD	XOR	CMP

(12)

For simplicity, let us also assume that the distinct opcodes in the metamorphic family of viruses are restricted to 5. The opcodes in the decreasing order of their frequencies are

MOV	ADD	PUSH	POP	CALL

Suppose that we are given an unknown file with the following opcode sequences

JMP
MOV
MOV
ADD
INC
INC
INC

(13)

The monograph statistics corresponding to the unknown file in (13) is

INC	MOV	ADD	JMP
3	2	1	1

The initial putative key K is

Metamorphic family of virus files:	MOV	ADD	PUSH	POP	CALL	(14)
Unknown file:	INC	MOV	ADD	JMP		

The content of the unknown file according to the key in (14) is

POP
ADD
ADD
PUSH
MOV
MOV
MOV

(15)

The digraph distribution matrix corresponding to the key in (15) is

	MOV	CALL	ADD	XOR	CMP	UNKNOWN
MOV	2	0	0	0	0	0
CALL	0	0	0	0	0	0
ADD	0	0	1	0	0	1
XOR	0	0	0	0	0	0
CMP	0	0	0	0	0	0
UNKNOWN	1	0	1	0	0	0

(16)

As in Jackobsen's approach, the first step in hill climbing is to swap the first two opcodes in (12). This is done by swapping the first two rows and columns of the digraph distribution matrix in (16). The modified matrix is given by

	MOV	CALL	ADD	XOR	CMP	UNKNOWN
MOV	0	0	0	0	0	0
CALL	0	2	0	0	0	0
ADD	0	0	1	0	0	1
XOR	0	0	0	0	0	0
CMP	0	0	0	0	0	0
UNKNOWN	0	1	1	0	0	0

(17)

It is clearly evident from the matrices (16) and (17) that, swapping the elements in the key is done by simply swapping the corresponding rows and columns of the D matrix, that begin or end with the swapped elements, which is the same as in Jackobsen's algorithm [6].

CHAPTER 6

Experiments

This section shows the results obtained by implementing the algorithm in Java. "Objdump" was the tool used to extract the opcode sequences from the executable files. The effectiveness of the technique is evaluated by measuring the similarity between the opcode sequences extracted from different metamorphic family of viruses and the benign executable files. The family of metamorphic viruses used for testing include Next Generation Virus Generation Kit (NGVCK), Second Generation Virus Generator (G2) and Metamorphic Worm (MWOR) generated by metamorphic generator in [16]. Cygwin utility files and Linux library files were used as benign files.

In [16], a metamorphic generator was developed for the sole purpose of defeating Hidden Markov Model (HMM) detection technique. The metamorphic generator uses two morphing techniques to achieve its goal:

- Garbage instruction insertion
- Equivalent instruction substitution

The viruses produced by this generator look different from each other across generations. And they look very much similar to benign files since blocks of dead code from one or more benign files are inserted directly into the virus files. The metamorphic generator produces virus with different "padding ratios". Padding ratio is defined as the ratio of number of dead code instructions to the number of instructions that constitute the core functionality of the virus. Padding ratio of 0.5 indicates that the virus has half as much dead code instructions as virus's instructions [16].

6.1 Test Data

Our test set of metamorphic viruses consists of 50 NGVCK files, 50 G2 files and 100 MWOR files with padding ratios of 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 and 4.0 (MWOR_0_5 ... MWOR_4_0). As the data set is relatively small, we used five-fold cross-validation technique in order to get the best out of it. In this technique, 80% of the data will be used for training and 20% of the data will be used for testing and the experiment will be repeated five times.

Our test set of benign files consists of 10 Cygwin Utility files and 20 Linux library files. The metamorphic generator in [16] uses these Linux library files to extract the dead code and insert it directly into the virus files.

We trained the detection system on a set of NGVCK family viruses and then scored against a different set of viruses under the same family and a set of Cygwin files. We conducted a similar experiment with G2 family viruses. We also trained the detection system on a set of MWOR with different padding ratios and scored against a different set of MWOR files and a set of Linux library files from which the dead code was extracted.

6.2 Test Results

The similarity scores obtained by comparing various metamorphic family of viruses with the virus files under the same family and benign executable files is shown in this section. We used Receiver Operating Characteristic (ROC) curve for evaluating the performance of the detection system. It is a two-dimensional graph, in which the X-axis represents the false positive rate and the Y-axis represents the true positive rate [22]. Accuracy of the detection system is measured by the area under the ROC curve (AUC) [16]. An AUC of 1 represents a perfect system; an area of 0.5 represents a worst system. In the remainder of this section, we summarize the results

of the tests that were conducted.

The graphs in Figures 4 and 5 show the similarity scores obtained by training the detection system on a set of NGVCK or G2 family of viruses respectively. Results in the graphs indicate that there is a clear separation between NGVCK or G2 family of viruses and the benign files using this technique.

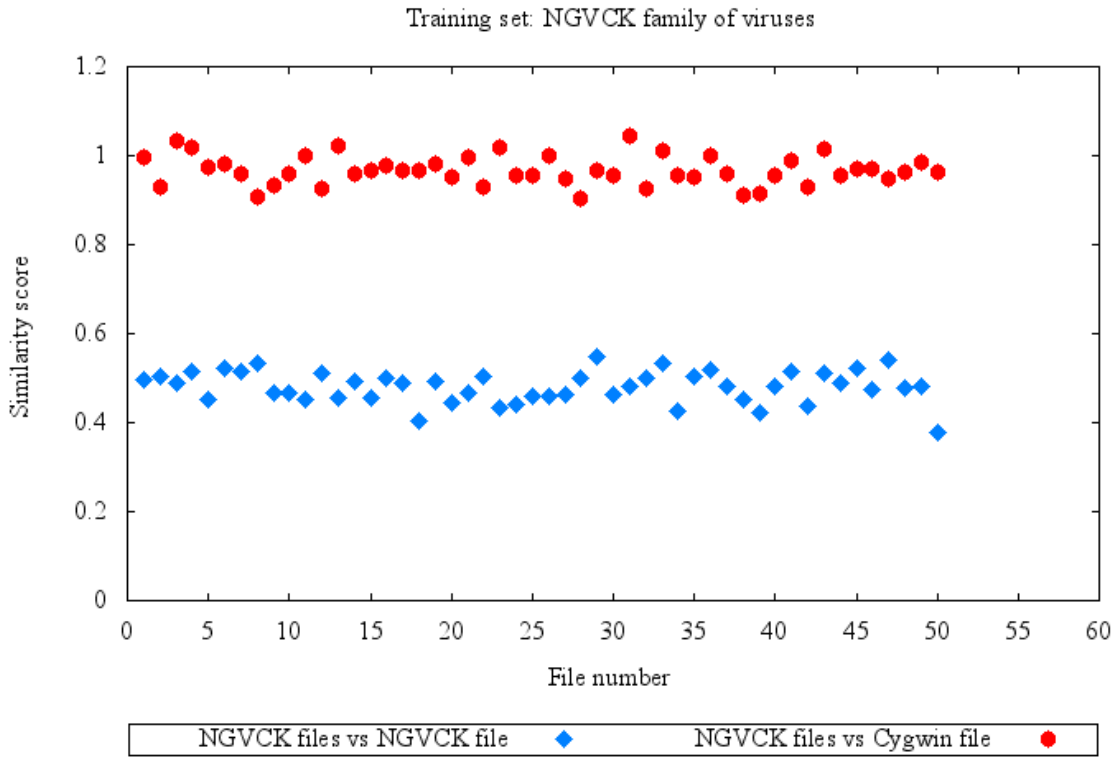

Figure 4: Training Set: NGVCK

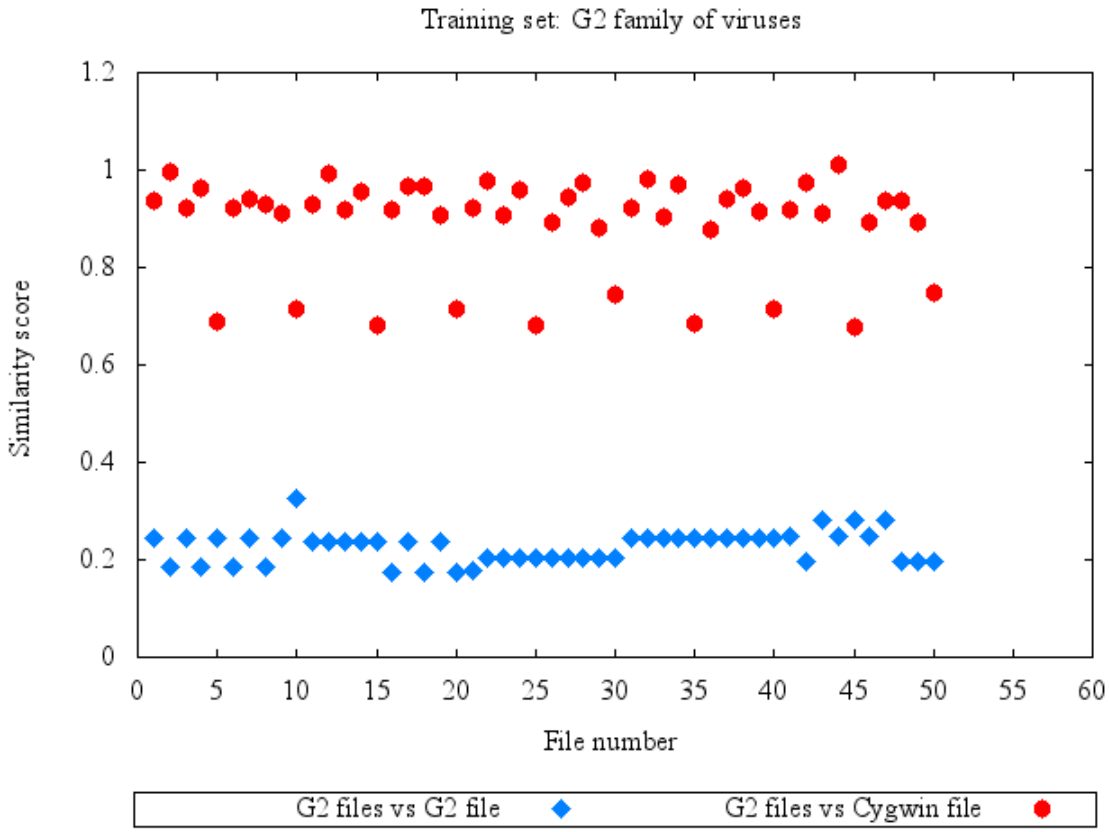

Figure 5: Training Set: G2

The similarity scores obtained by training the detection system on a set of MWOR files with different padding ratios are shown by the following graphs. Results indicate that as more code is copied from the benign files into the virus files, closer the scores approach to that of the benign files. When the padding ratio is greater than or equal to 1.5, there are lots of misclassifications. The performance of detection system trained on MWOR with different padding ratios is illustrated by the ROC curves in Figure 14. The AUC and standard error for each of the curves in the graph is shown in Table 4.

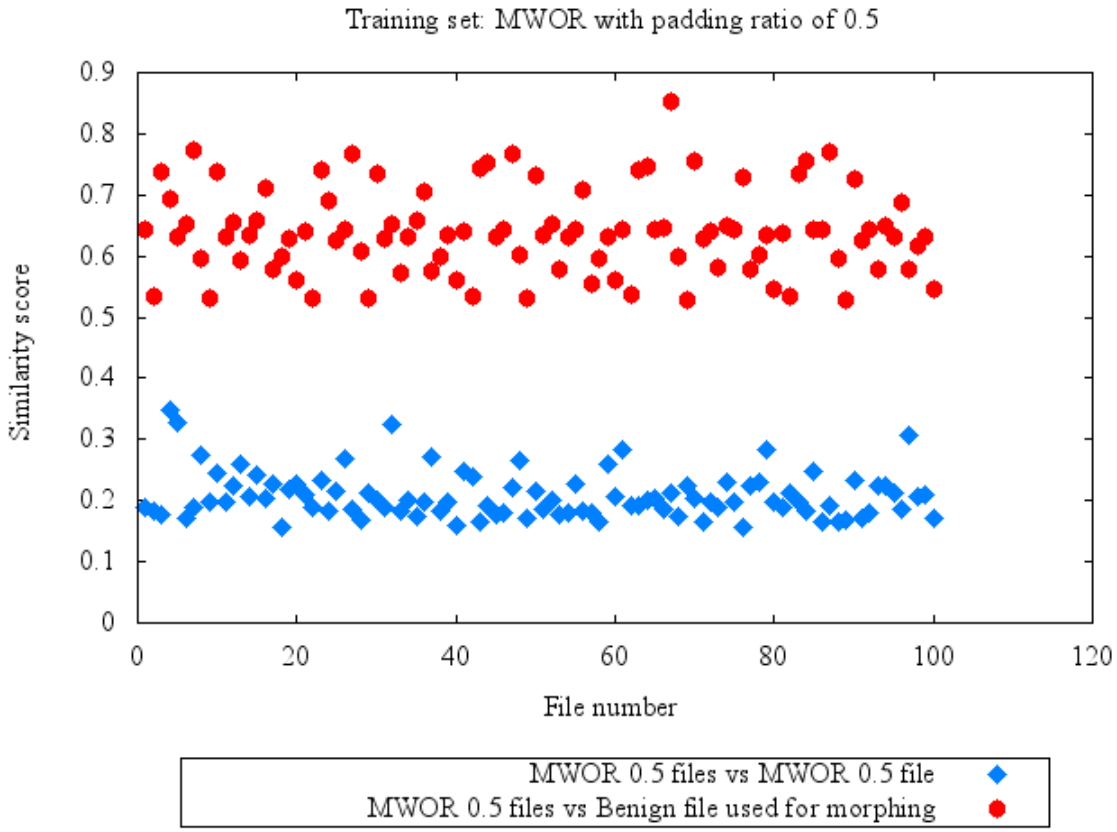

Figure 6: Training Set: MWOR with Padding Ratio of 0.5

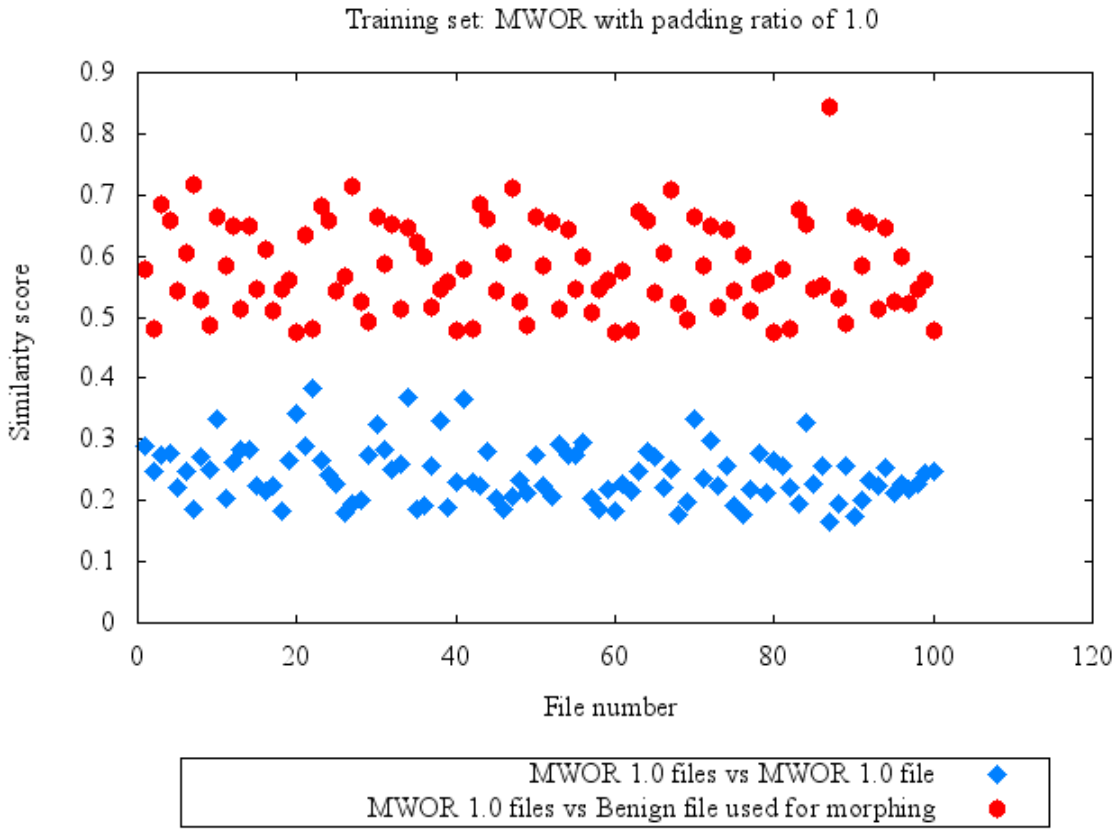

Figure 7: Training Set: MWOR with Padding Ratio of 1.0

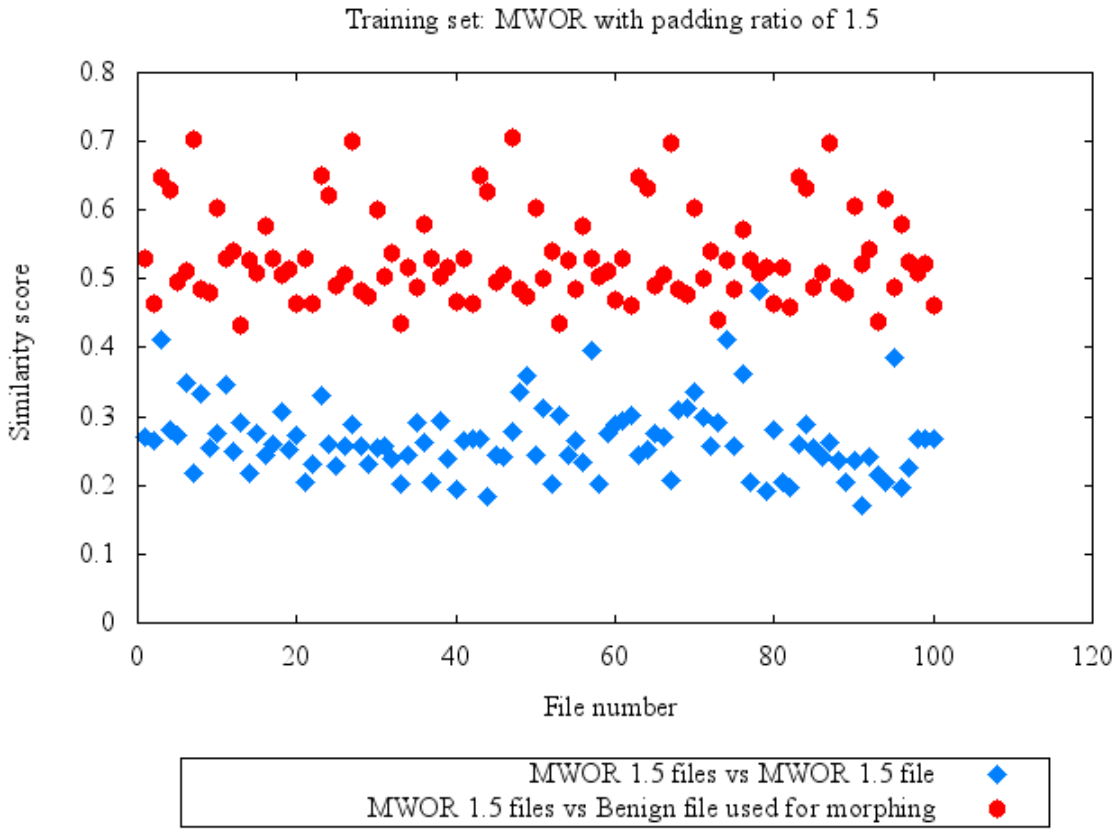

Figure 8: Training Set: MWOR with Padding Ratio of 1.5

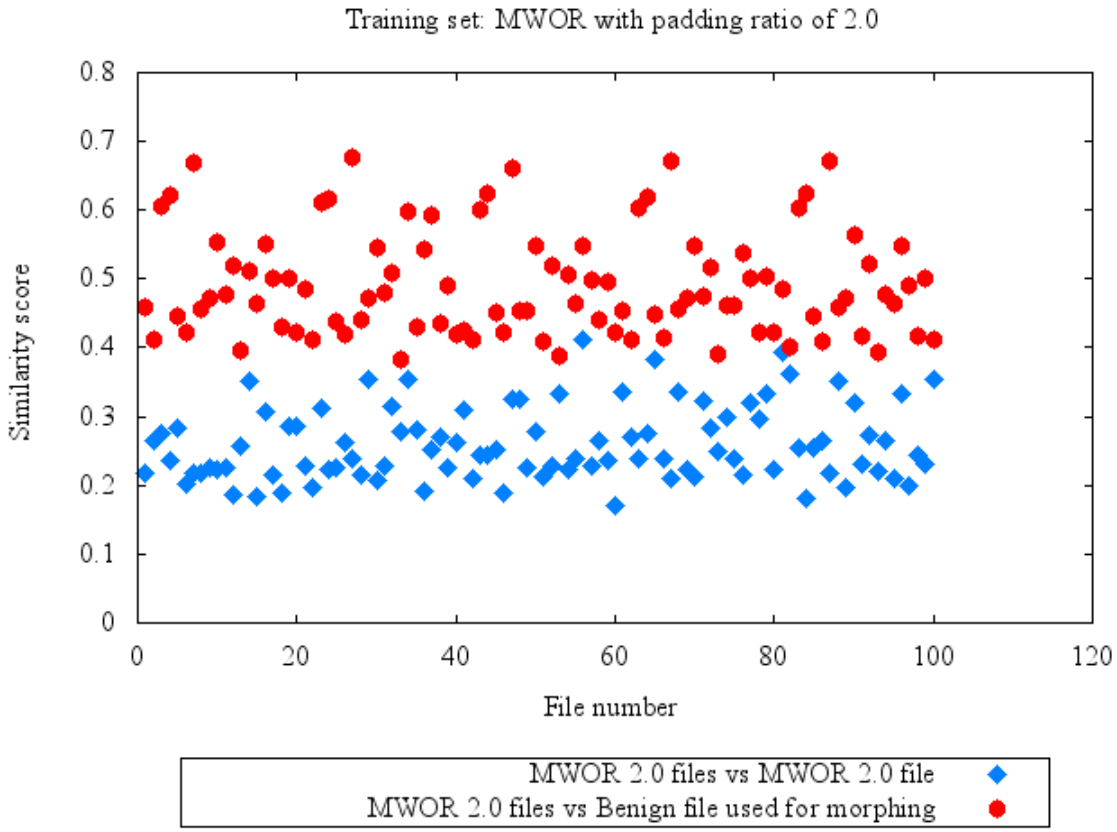

Figure 9: Training Set: MWOR with Padding Ratio of 2.0

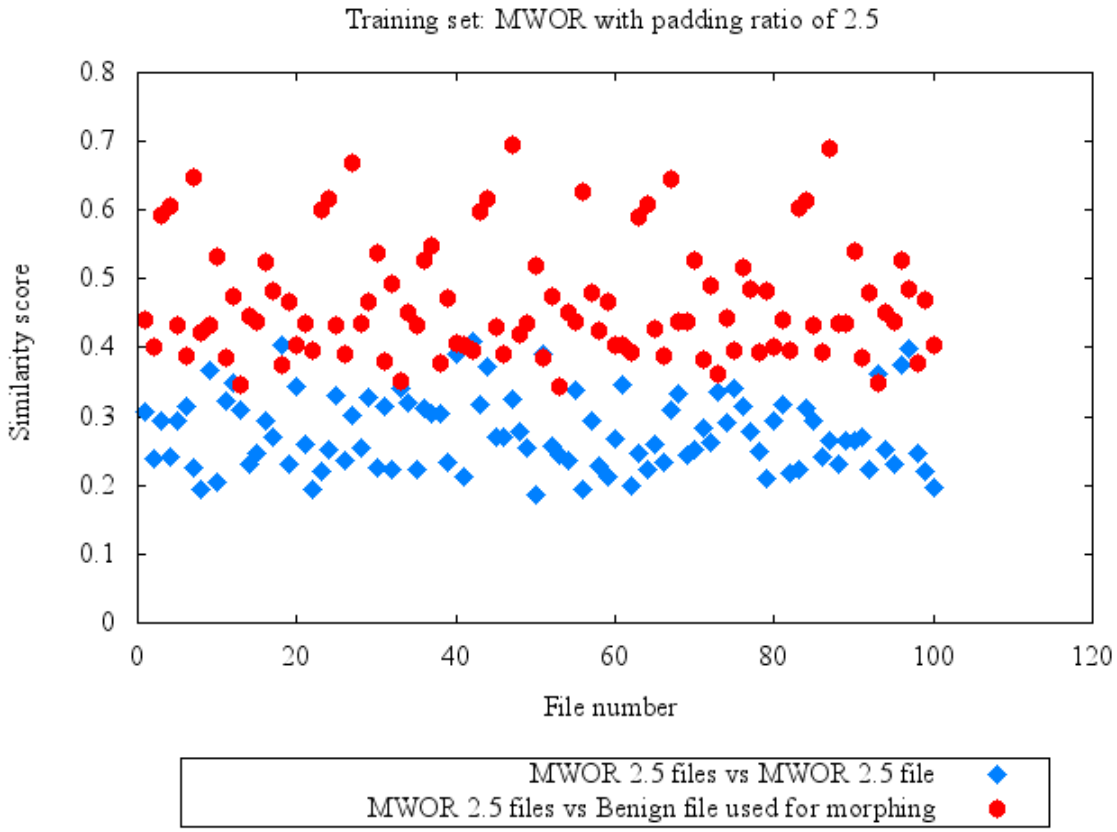

Figure 10: Training Set: MWOR with Padding Ratio of 2.5

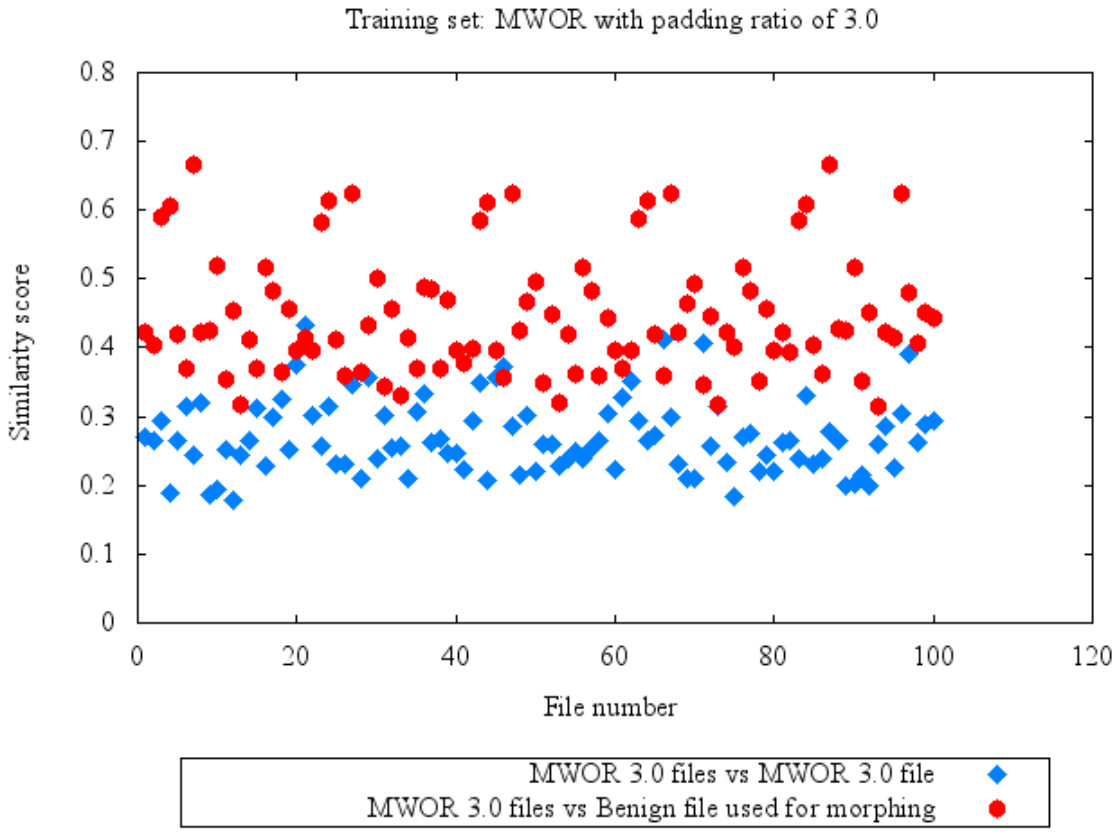

Figure 11: Training Set: MWOR with Padding Ratio of 3.0

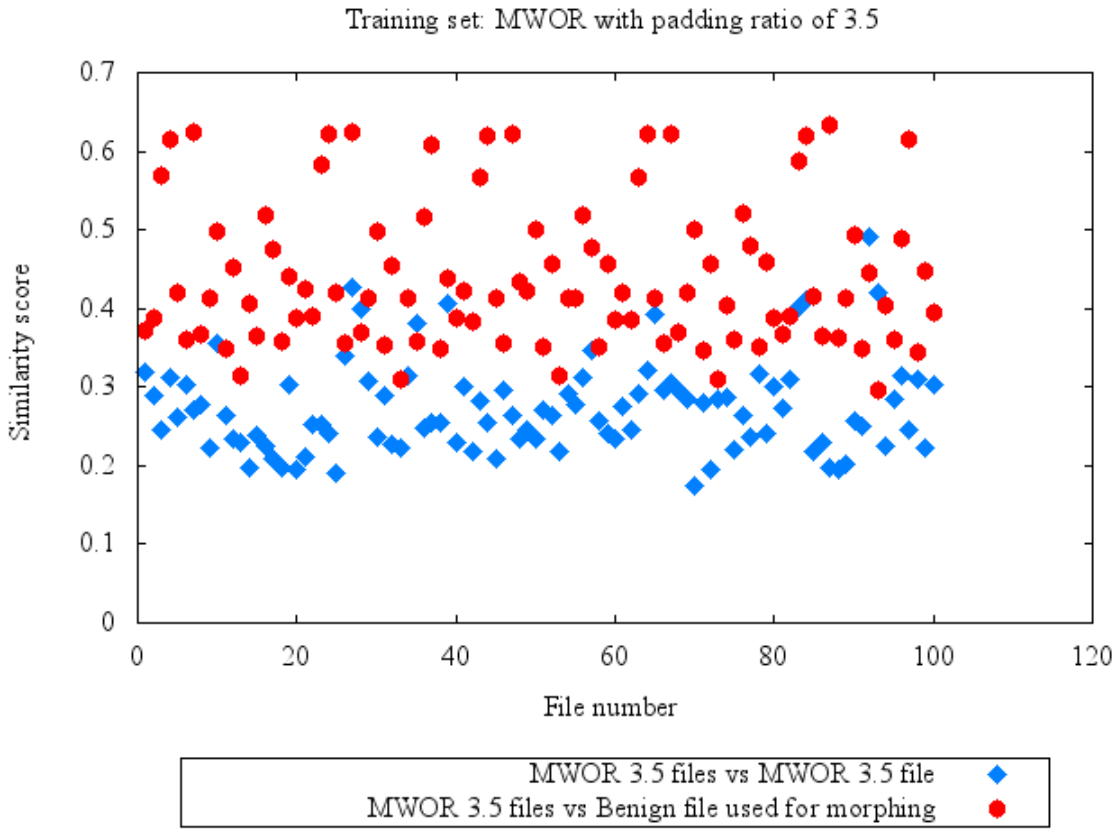

Figure 12: Training Set: MWOR with Padding Ratio of 3.5

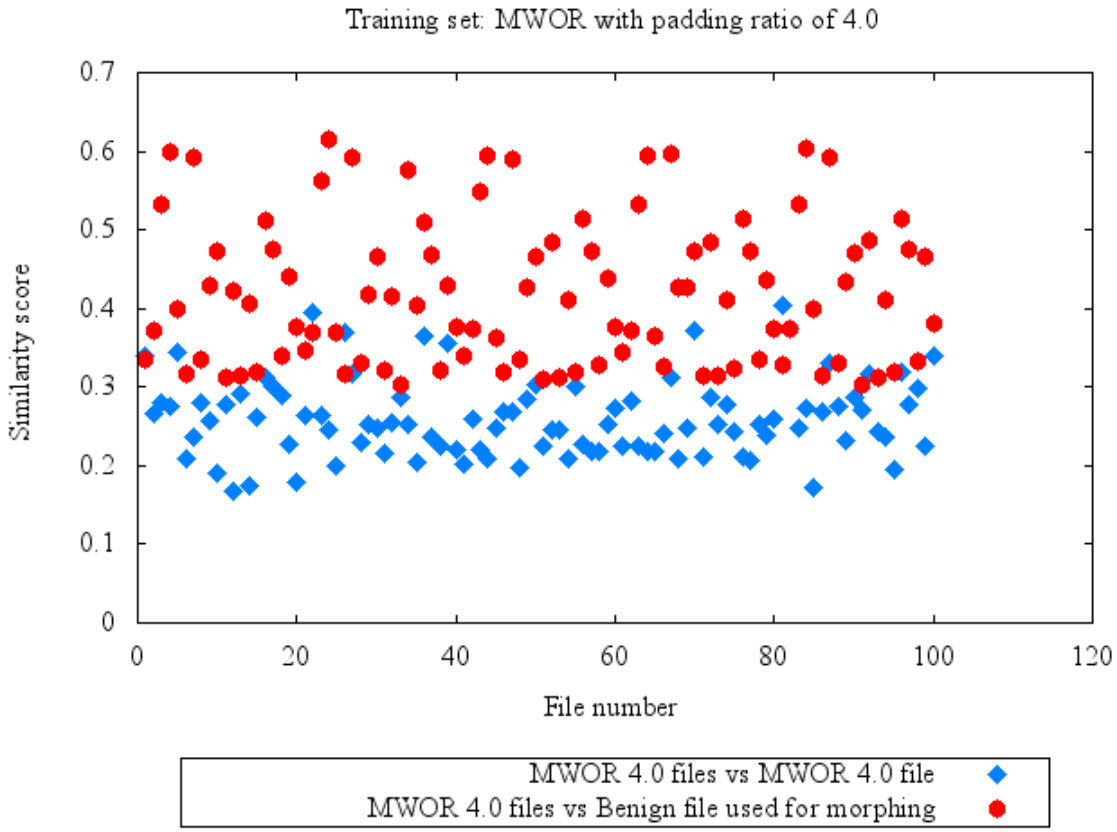

Figure 13: Training Set: MWOR with Padding Ratio of 4.0

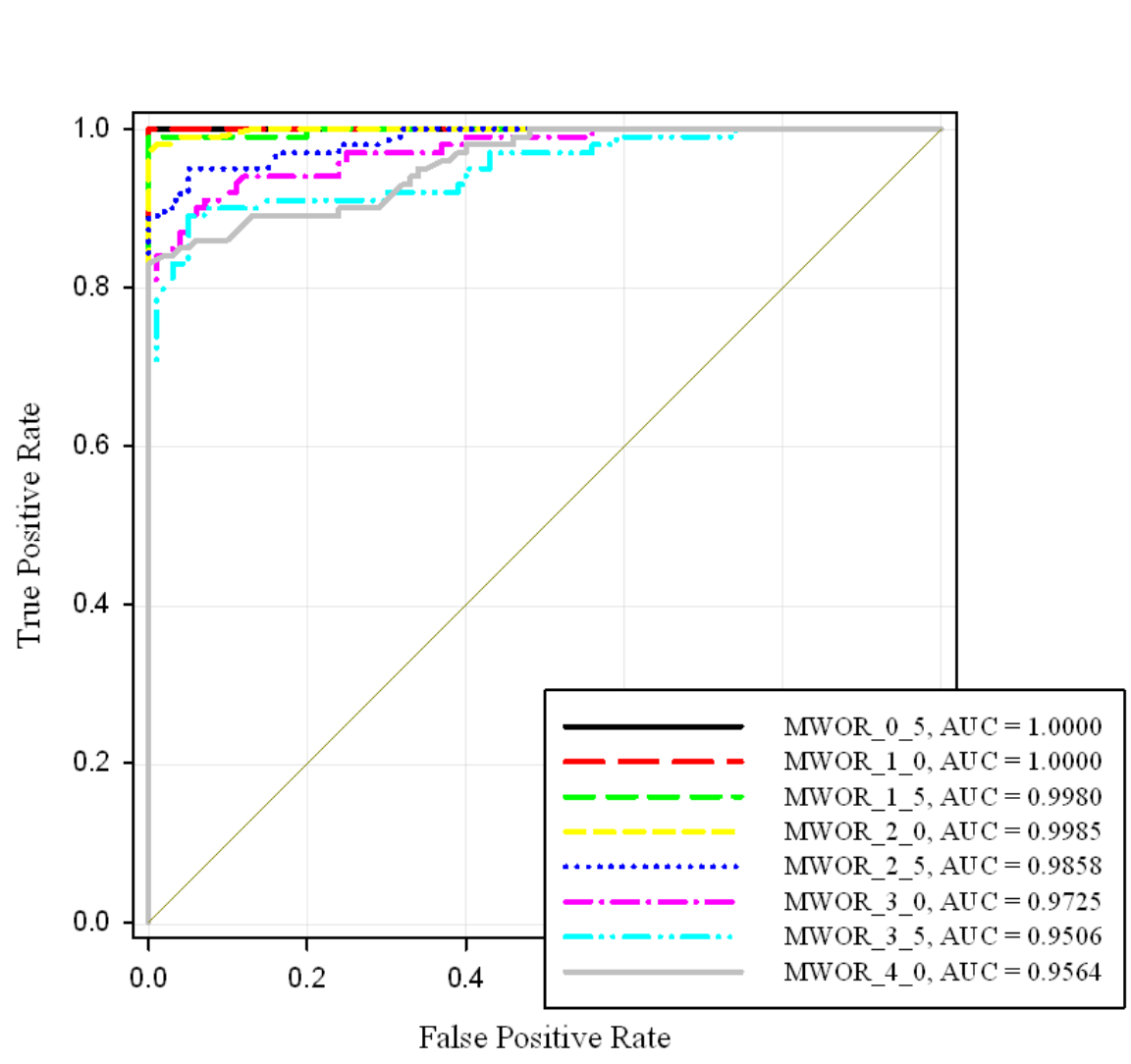

Figure 14: ROC Curves for Different Padding Ratios

Table 4: ROC AUC Statistics for Different Padding Ratios

Padding-ratio	AUC	Standard Error
0.5	1	0
1.0	1	0
1.5	0.998	0.00207
2.0	0.9985	0.00129
2.5	0.9858	0.00605
3.0	0.9725	0.00974
3.5	0.9506	0.01502
4.0	0.9564	0.01277

6.2.1 Different Scoring Functions Experimented

In order to determine the effective scoring function for evaluating the goodness of the putative key K, we experimented with the following six scoring functions and analyzed their strengths. We conducted experiments using MWOR with padding ratio of 4.0. The results are summarized in the ROC curves shown in Figure 15. The AUC and standard error for each of the curves in the graph is shown in Table 5. ROC curves in Figure 15 show that $score_1$ and $score_2$ perform better than the other scoring functions.

$$score_1(K) = d(D, E) = \sum_{i,j} |d_{ij} - e_{ij}|$$

$$score_2(K) = d(D, E) = \frac{1}{n^2} \sum_{i,j} |d_{ij} - e_{ij}|$$

$$score_3(K) = d(D, E) = \sum_{i,j} |d_{ij} - e_{ij}|^2$$

$$score_4(K) = d(D, E) = \frac{1}{n^2} \sum_{i,j} |d_{ij} - e_{ij}|^2$$

$$score_5(K) = d(D, E) = \sum_{i,j} |d_{ij}^2 - e_{ij}^2|$$

$$score_6(K) = d(D, E) = \frac{1}{n^2} \sum_{i,j} |d_{ij}^2 - e_{ij}^2|$$

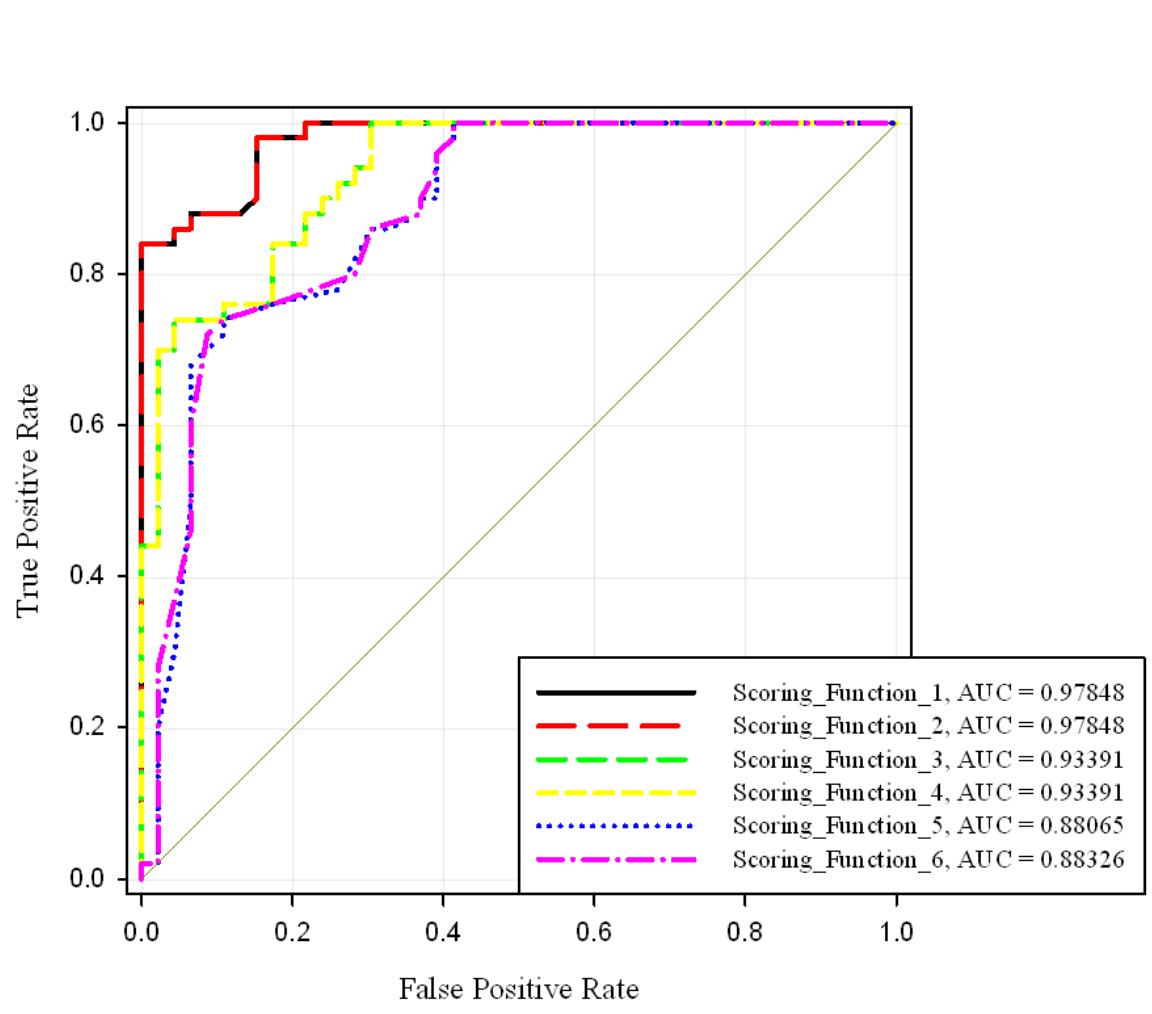

Figure 15: ROC Curves for Different Scoring Functions

Table 5: ROC AUC Statistics for Different Scoring Functions

Scoring Systems	AUC	Standard Error
Scoring System 1	0.97848	0.01073
Scoring System 2	0.97848	0.01073
Scoring System 3	0.93391	0.02287
Scoring System 4	0.93391	0.02287
Scoring System 5	0.88065	0.03501
Scoring System 6	0.88326	0.03436

In order to choose between $score_1$ and $score_2$, we conducted experiments using

MWOR with padding ratio of 3.5. Since $score_1$ and $score_2$ produced similar results in that test setup too, we used $score_1$ for scoring the putative key K in this technique.

6.2.2 Different Size of the Matrix Experimented

For a particular processor, there are more than 130 opcodes [19]. If we allocate a row/column for all of the opcodes, then the size of the matrices will become too large to compare. Also the opcodes which occur less frequently or which does not occur in the family viruses or the benign files will not contribute much for the similarity score. So we parsed through around 600 different disassembled virus and benign files to collect the top n most frequent distinct opcodes. We conducted experiments with the following different values to determine an effective value for n.

```
15,20,25,30,35,50,60,100
```

For the experiments, we trained the system using MWOR with padding ratio of 4.0. Results are summarized in the ROC curves shown in Figure 16. The AUC and standard error for each of the curves in the graph is shown in Table 6. ROC curves in Figure 16 show that $n = 25$ works better for this technique. Hence we created D and E matrices of size (26×26) for this technique. Experiments show that AUC remains the same for value of n beyond 50. This is because those opcodes occur less frequently or not at all in the metamorphic family of viruses or the benign files and they do not contribute anything for the similarity score.

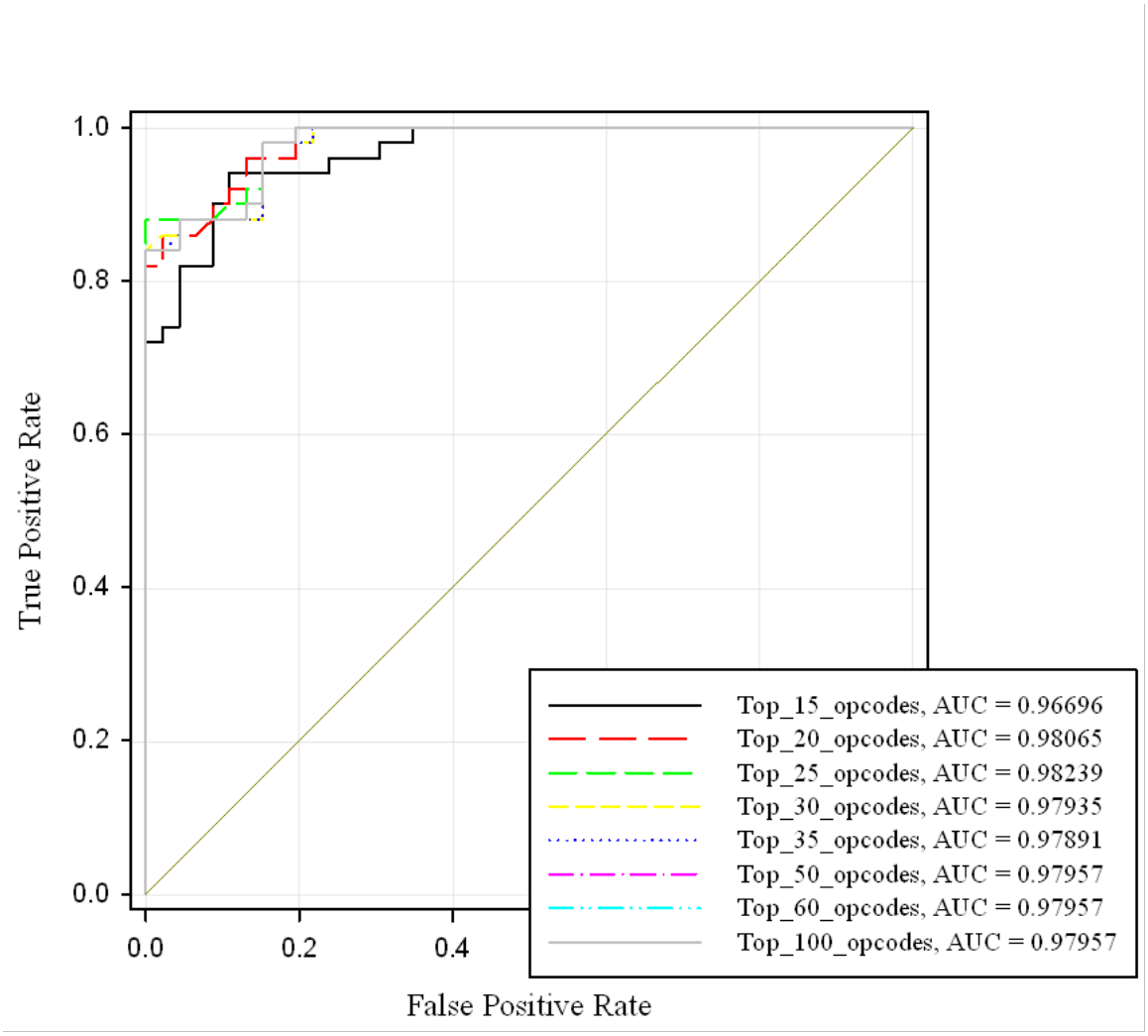

Figure 16: ROC Curves for Different Values of n

Table 6: ROC AUC Statistics for Different Values of n

Different n values	AUC	Standard Error
15	0.96696	0.01471
20	0.98065	0.00989
25	0.98239	0.00944
30	0.97935	0.01052
35	0.97891	0.01064
50	0.97957	0.01035
60	0.97957	0.01035
100	0.97957	0.01035

6.2.3 Different Normalization Techniques Experimented

We experimented with two different normalization techniques for normalizing the E matrix. Let m denote the set of metamorphic family of virus files, on which the detection system is trained on.

- **Normalization Technique 1**

  We parse through all the m files and construct an E matrix of size (26×26). This matrix contains the sum of digraph distributions of opcodes in all the m files. We then normalize the E matrix by dividing each cell in the matrix by the sum of all the cells in the matrix.

- **Normalization Technique 2**

  We construct m matrices of size (26×26). matrix 0 contains the digraph distribution of opcodes in file 0, matrix 1 contains the digraph distribution of opcodes in file 1 and so on. We normalize the matrices matrix 0 to matrix $m-1$ by dividing each cell in the matrix by the sum of all the cells in the matrix.

  We then construct E matrix as follows.

$$E = \{e_{ij}\} = (matrix0_{ij} + matrix1_{ij} + ... + matrix(m-1)_{ij})/m \qquad (18)$$

We conducted the experiment using MWOR with padding ratio of 4.0. ROC curves in Figure 17 summarize the results obtained using the two techniques. The AUC and standard error for each of the curves in the graph is shown in Table 7. ROC curves in Figure 17 show that technique 2 performs better than the technique 1. Hence we used technique 2 for normalizing the E matrix.

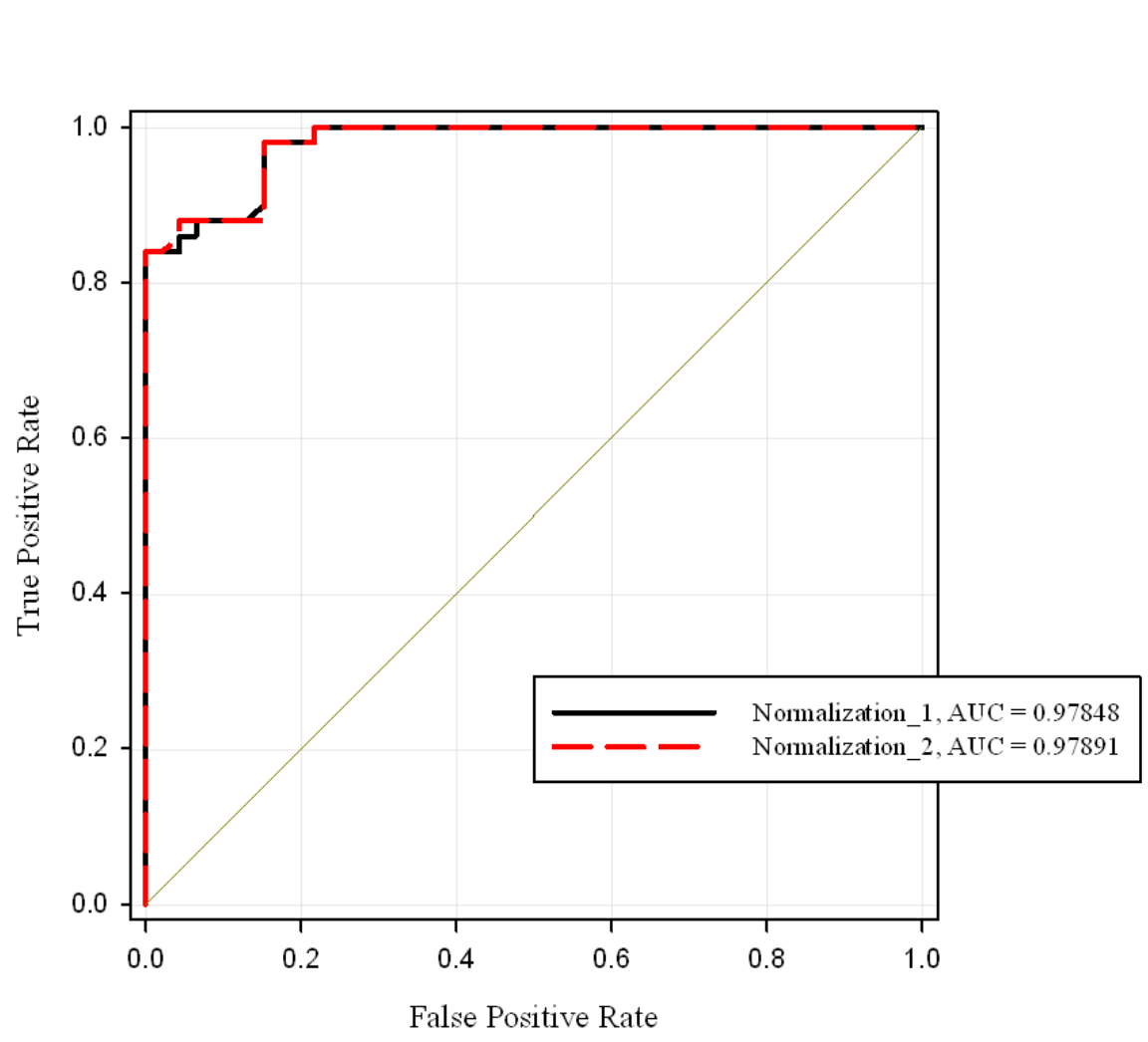

Figure 17: ROC Curves for Normalization Techniques 1 and 2

Table 7: ROC AUC Statistics for Normalization Techniques 1 and 2

Normalization Techniques	AUC	Standard Error
Normalization Technique 1	0.97848	0.01073
Normalization Technique 2	0.97891	0.01064

6.2.4 Different Swapping Strategies Experimented

We experimented with the following six strategies to determine an effective swapping strategy for this technique. For the experiments, we trained the system using

MWOR with padding ratio of 4.0. Results obtained for different swapping strategies are summarized in the ROC curves in Figure 18. The AUC and standard error for each of the curves in the graph is shown in Table 8. ROC curves in Figure 18 show that the swapping strategy 2 works better for this technique. Hence we used that as the swapping strategy for this technique.

- **Swapping Strategy 1**

  In this technique, we swap all adjacent pairs of opcodes, then all pairs at distance of 2, then all pairs at distance of 3 and so on until we complete exactly $\binom{n}{2}$ swaps.

- **Swapping Strategy 2**

  In this technique, we swap as in Strategy 1, but any time the score improves, we start again from the beginning.

- **Swapping Strategy 3**

  This is another variant of Strategy 1. In this case, we swap all adjacent pairs of opcodes, then all pairs at distance 2, then add pairs at distance 3, and so on. Once we complete the $\binom{n}{2}$ steps, we iterate the entire process, repeating until we complete one entire iteration without any swap improving the score.

- **Swapping Strategy 4**

  In this technique, we only swap adjacent pairs of opcodes. That is, we make only n swaps.

- **Swapping Strategy 5**

  This is similar to Strategy 4, except that whenever a swap improves the score, we continue swapping that element until the score no longer improves, at which point we revert to the position where the series of swaps began.

- **Swapping Strategy 6**

  As in Strategy 4, we do n swaps of adjacent pairs. We then repeat, until we go through one entire iteration without any swap improving the score.

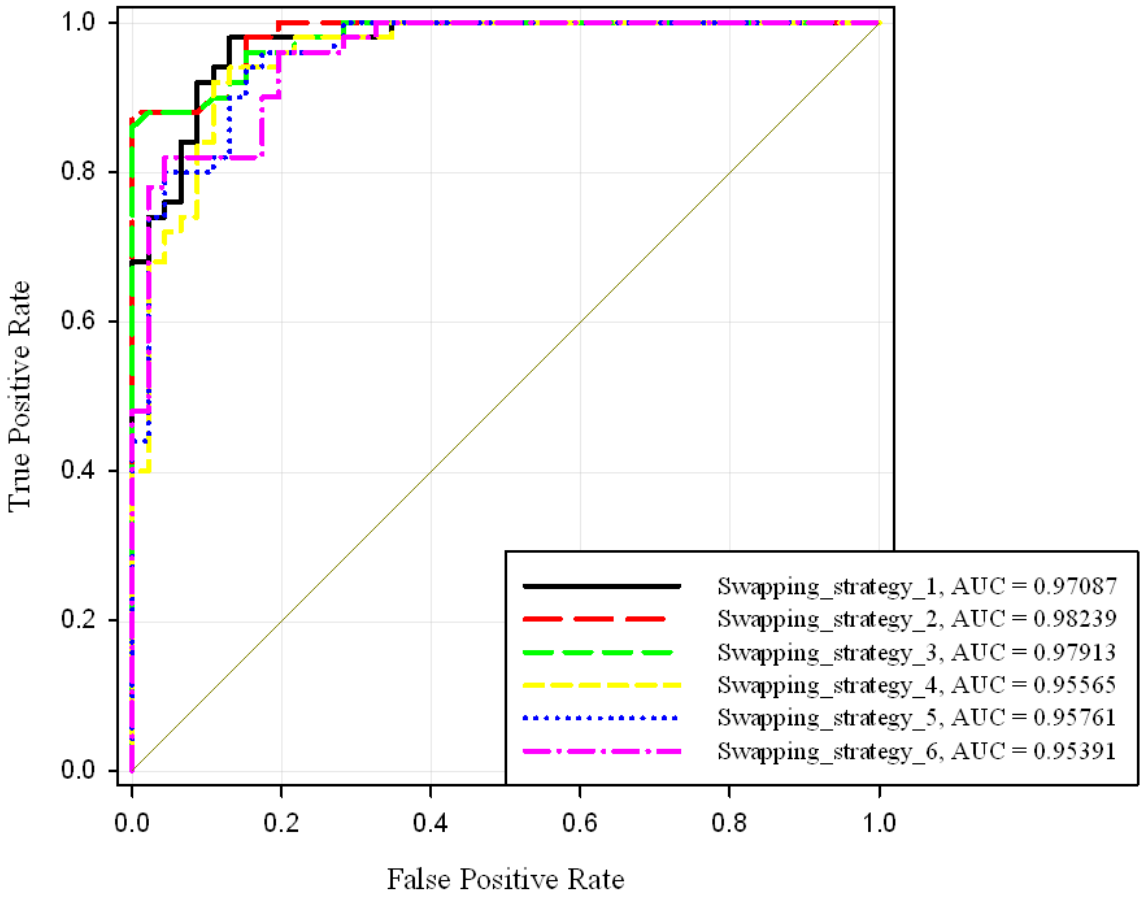

Figure 18: ROC Curves for Different Swapping Strategies

Table 8: ROC AUC Statistics for Different Swapping Strategies

Different swapping strategies	AUC	Standard Error
Swapping strategy 1	0.97087	0.01373
Swapping strategy 2	0.98239	0.00944
Swapping strategy 3	0.97913	0.01080
Swapping strategy 4	0.95565	0.01883
Swapping strategy 5	0.95761	0.01782
Swapping strategy 6	0.95391	0.01850

6.2.5 Comparison with HMM Detection Technique

Hidden Markov Model (HMM) detection technique was tested in [16] by training the model using MWOR with different padding ratios. Comparing our results with the results obtained using HMM technique [16], we see that our technique performs better. Here are the ROC AUC statistics for both the techniques.

Padding ratio	AUC	Standard Error
0.5	1	0
1.0	1	0
1.5	0.998	0.00207
2.0	0.9985	0.00129
2.5	0.98585	0.00605
3.0	0.9725	0.00974
4.0	0.95645	0.01277

Table 9: ROC AUC Statistics for Different Padding Ratios using Proposed Technique

Padding ratio	AUC	Standard Error
0.5	1	0
1.0	0.99	0.0105
1.5	0.9625	0.03503
2.0	0.9725	0.02112
2.5	0.8325	0.06556
3.0	0.8575	0.06225
4.0	0.8225	0.06661

Table 10: ROC AUC Statistics for Different Padding Ratios using HMM Detection Technique [16]

6.2.6 Efficiency of the Proposed Technique

Efficiency of the proposed technique can be measured in terms of number of score computations, file size, number of swaps it requires and the time it takes to compute the similarity score. Table 11 shows the score computation count, swap count, file

size and the time taken to compute the similarity score (in milliseconds), when the scoring technique is trained using MWOR with different padding ratios.

Table 11: Scoring Efficiency for MWOR Family Viruses and Benign Files

Comparison	No. of Score Computations	Swap Count	Average Time (in milliseconds)	File Size (in kilobytes)
MWOR 0.5 family viruses vs MWOR 0.5 virus	Average: 1584 Minimum: 603 Maximum: 3704	Average: 23 Minimum: 6 Maximum: 63	29.5	20.95
MWOR 0.5 family viruses vs Benign file	Average: 1831 Minimum: 1067 Maximum: 2592	Average: 32 Minimum: 20 Maximum: 48	36.2	84.6
MWOR 1.0 family viruses vs MWOR 1.0 virus	Average: 1251 Minimum: 492 Maximum: 2145	Average: 23 Minimum: 5 Maximum: 35	26.45	27.25
MWOR 1.0 family viruses vs Benign file	Average: 1827 Minimum: 1167 Maximum: 2785	Average: 37 Minimum: 19 Maximum: 55	37.45	84.6
MWOR 1.5 family viruses vs MWOR 1.5 virus	Average: 1114 Minimum: 585 Maximum: 1824	Average: 23 Minimum: 11 Maximum: 46	24	34.25
MWOR 1.5 family viruses vs Benign file	Average: 1387 Minimum: 922 Maximum: 1929	Average: 31 Minimum: 19 Maximum: 42	36	84.6
MWOR 2.0 family viruses vs MWOR 2.0 virus	Average: 924 Minimum: 660 Maximum: 1597	Average: 19 Minimum: 10 Maximum: 37	23.85	41
MWOR 2.0 family viruses vs Benign file	Average: 1493 Minimum: 942 Maximum: 2220	Average: 34 Minimum: 22 Maximum: 55	34.6	84.6
MWOR 2.5 family viruses vs MWOR 2.5 virus	Average: 1023 Minimum: 591 Maximum: 1679	Average: 21 Minimum: 11 Maximum: 33	31.15	48.05
MWOR 2.5 family viruses vs Benign file	Average: 1586 Minimum: 868 Maximum: 2940	Average: 37 Minimum: 18 Maximum: 65	35.55	84.6
MWOR 3.0 family viruses vs MWOR 3.0 virus	Average: 1118 Minimum: 744 Maximum: 1646	Average: 24 Minimum: 16 Maximum: 35	27.2	55
MWOR 3.0 family viruses vs Benign file	Average: 1529 Minimum: 988 Maximum: 2799	Average: 35 Minimum: 25 Maximum: 48	34.25	84.6
MWOR 3.5 family viruses vs MWOR 3.5 virus	Average: 1114 Minimum: 674 Maximum: 1920	Average: 24 Minimum: 11 Maximum: 51	25.25	58.85
MWOR 3.5 family viruses vs Benign file	Average: 1612 Minimum: 830 Maximum: 3055	Average: 38 Minimum: 24 Maximum: 67	35.9	84.6
MWOR 4.0 family viruses vs MWOR 4.0 virus	Average: 1020 Minimum: 533 Maximum: 1524	Average: 21 Minimum: 8 Maximum: 32	23.3	68.7
MWOR 4.0 family viruses vs Benign file	Average: 1588 Minimum: 955 Maximum: 2426	Average: 36 Minimum: 23 Maximum: 53	35.2	84.6

CHAPTER 7

Conclusions and Future Work

We designed and implemented an opcode-based software similarity technique for metamorphic malware detection. The algorithm uses a hill climb approach analogous to the one used in simple substitution cipher attack [6]. The algorithm was implemented and extensively tested. Results have shown that we can easily set a threshold that clearly separates NGVCK or G2 family viruses and the benign files.

The proposed technique also achieved an accuracy rate of 100% when we tested it against MWOR, provided those padding ratios are 1.0 and below. The probability of misclassification starts increasing for padding ratios beyond 1.0, as indicated by the ROC curves in Figure 14. Results clearly indicate that the proposed technique performs very well when "equivalent instruction substitution" is used as the morphing technique, while it actually yields worse results when blocks of code from benign files are used for morphing. However, it is interesting to note that, although the accuracy rate in detecting MWOR with padding ratio of 1.5 and above is not 100%, this technique performs better than HMM detection technique which is shown in section 6.2.5.

Previous research has shown that similarity technique based on chi-squared distance had performed better in detecting metamorphic viruses if "subroutine insertion" was used as the morphing technique [22]. It might be useful to create a hybrid model using the proposed technique and the chi-squared distance technique to produce a stronger metamorphic virus detector.

LIST OF REFERENCES

[1] J. Aycock, *Computer Viruses and Malware*, Springer, 2006

[2] J. Borello and L. Me, Code Obfuscation Techniques for Metamorphic Viruses, *Journal in Computer Virology*, Vol. 4, No. 3, pp. 30-40, 2008

[3] Cygwin, Cygwin Utility Files,
http://www.cygwin.com/

[4] S. Govindaraj, Practical Detection of Metamorphic Computer Viruses, Master's report, Department of Computer Science, San Jose State University, 2008,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=1092&context=etd_projects

[5] N. Idika and A. Mathur, A Survey of Malware Detection Techniques, Technical report, Department of Computer Science, Purdue University, 2007,
http://www.serc.net/system/files/SERC-TR-286.pdf

[6] T. Jakobsen, A Fast Method for the Cryptanalysis of Substitution Ciphers, *Cryptologia*, Vol.19, pp. 265 – 274, 1995

[7] D. Lin, Hunting for Undetectable Metamorphic Viruses, Master's report, Department of Computer Science, San Jose State University, 2009,
http://scholarworks.sjsu.edu/etd_projects/18/

[8] J. Mathai, History of Computer Cryptography and Secrecy System,
http://www.dsm.fordham.edu/~mathai/crypto.html

[9] I. Muttik, Silicon Implants, *Virus Bulletin*, pp. 8–10, May 1997

[10] C. Nachenberg, Understanding and Managing Polymorphic viruses, *The Symantec Enterprise Papers*, Vol. 30,
http://www.symantec.com/avcenter/reference/striker.pdf

[11] C. Nachenberg, Understanding Heuristics: Symantec's Bloodhound Technology, *Symantec White Paper Series*, Vol. 34,
http://www.symantec.com/avcenter/reference/heuristc.pdf

[12] D. Oranchak, Evolutionary Algorithm for Decryption of Monoalphabetic Homophonic Substitution Ciphers Encoded as Constraint Satisfaction Problems, Technical report, NTU School of Engineering and Applied Science, 2008,
http://oranchak.com/t14pap379-oranchak.pdf

[13] M. Patel, Similarity Tests for Metamorphic Virus Detection, Master's report, Department of Computer Science, San Jose State University, 2011,
http://www.cs.sjsu.edu/faculty/stamp/students/
patel_mahim.pdf

[14] S. Priyadarshi, Metamorphic Detection via Emulation, Master's report, Department of Computer Science, San Jose State University, 2011,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=1176&context=etd_projects

[15] B. B. Rad, M. Masrom, S. Ibrahim, Evolution of Computer Virus Concealment and Anti-Virus Techniques: A Short Survey, *IJCSI International Journal of Computer Science Issues*, Vol.8, Issue 1, January 2011,
http://arxiv.org/pdf/1104.1070.pdf

[16] S. M. Sridhara, Metamorphic Worm that carries its own Morphing Engine, Master's report, Department of Computer Science, San Jose State University, 2012,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=1238&context=etd_projects

[17] M. Stamp, *Information Security: Principles and Practice*, second edition, Wiley, 2011

[18] M. Stamp, R. M. Low and A. Dhavare, Efficient Cryptanalysis of Homophonic Substitution Ciphers, Technical report, Department of Computer Science, San Jose State University, 2011,
http://www.cs.sjsu.edu/faculty/stamp/RUA/
homophonic.pdf

[19] M. Stamp, R. M. Low and N. Runwal, Opcode Graph Similarity and Metamorphic Detection, *Journal in Computer Virology*, Vol. 8, Nos. 1-2, pp. 37-52, May 2012

[20] P. Szor, The Art of Computer Virus Research and Defense, First edition, Addison-Wesley, 2005,
http://computervirus.uw.hu/ch11lev1sec4.html

[21] P. Szor and P. Ferrie, Hunting for Metamorphic, Symantec Security Response,
http://www.symantec.com/avcenter/reference/
hunting.for.metamorphic.pdf

[22] A. H. Toderici, Chi-squared Distance and Metamorphic Virus Detection, Master's report, Department of Computer Science, San Jose State University, 2012,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=7710&context=etd_theses

[23] S. Venkatachalam, Detecting Undetectable Computer Viruses, Master's report, Department of Computer Science, San Jose State University, 2010,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=1155&context=etd_projects

[24] A. Venkatesan, Code Obfuscation and Virus Detection, Master's report, Department of Computer Science, San Jose State University, 2008,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=1115&context=etd_projects/

[25] Wikipedia, Computer Virus,
http://en.wikipedia.org/wiki/Computer_virus

[26] Wikipedia, Hill Climbing,
http://en.wikipedia.org/wiki/Hill_climbing

[27] Wikipedia, Substitution Cipher,
http://en.wikipedia.org/wiki/Substitution_cipher

[28] W. Wong, Analysis and Detection of Metamorphic Computer Viruses, Master's report, Department of Computer Science, San Jose State University, 2006,
http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?
article=1152&context=etd_projects