2006

Analysis and Detection of Metamorphic Computer Viruses

Wing Wong
San Jose State University

ANALYSIS AND DETECTION OF METAMORPHIC COMPUTER VIRUSES

A Writing 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
Wing Wong

May, 2006

Approved by: **Department of Computer Science**
College of Science
San Jose State University
San Jose, CA

_____

Dr. Mark Stamp

_____

Dr. Robert Chun

_____

Dr. Suneuy Kim

1. INTRODUCTION

"A computer virus is a program that recursively and explicitly copies a possibly evolved version of itself" [16]. A virus copies itself to a host file or system area. Once it gets control, it multiplies itself to form newer generations. A virus may carry out damaging activities on the host machine like corrupting or erasing files, overwriting the whole hard disk, or crashing the computer. Some viruses may print text on the screen or simply do nothing. These viruses remain harmless but keep reproducing themselves. In any case, they take up system resource and are undesirable to computer users.

Over the past two decades, the number of viruses has been increasing very rapidly. We have seen several attacks that caused great disruption to the Internet and brought huge damage to organizations and individuals. In 1999, the infamous Melissa virus infected thousands of computers and caused damage close to $80 million; the Code Red worm outbreak in 2001 affected systems running Windows NT and Windows 2000 server and caused damage in excess of $2 billion [20]. Computer virus attacks will continue to pose serious security threats to every computer user.

To simplify the virus creation process, virus writers make virus construction kits readily available on the Internet [19]. This allows a lot more people who may not have the expertise in assembly coding to generate their own viruses. These virus writers also recognize that for their viruses to have a chance to escape detection, the viruses created have to look substantially different from one another. Some kits come equipped with the

ability to generate automatically morphed variants even from a single configuration file. Precisely how effective are these code morphing generators? How different do the morphed variants look? We generated variants of a few metamorphic viruses using some of these tools and measured the similarity between some morphed variants.

Detecting metamorphic viruses is challenging. The problem with simple signature-based scanning is that even small changes in the viral code may cause a scanner to fail and the signature database requires constant updates to signify newly morphed variants. We experimented using a single hidden Markov model (HMM) to represent the behavior of a whole virus family. The HMM is then used to determine whether a given program belongs to the virus family that the HMM represents. This approach can be used to distinguish member viruses from non-member programs.

The challenges with the HMM approach include finding the right balance between sensitivity and specificity, and conforming to time and space constraints of the computers performing the detection. We evaluate the effectiveness of this approach by its detection rate, the amount of false positives and false negatives, and the overall accuracy of the classification.

This paper is organized as follows. In Section 2, we provide some background information on computer viruses and discuss some possible defenses. Section 3 describes our virus similarity test and shows our results. Section 4 details the design and

implementation of our HMM approach and presents our experimental results. Section 5 is our conclusion. And finally possible extension to the project and future work is discussed in Section 6.

2. EVOLUTION OF VIRUSES AND ANTIVIRUS DEFENSE TECHNIQUES

2.1 Virus Obfuscation Techniques

Virus-like programs first appeared on microcomputers in the 1980s [16]. Since then, the battle between virus writers and anti-virus (AV) researchers has never ceased. To challenge virus scanning products, virus writers constantly develop new obfuscation techniques to make virus code more difficult to detect [16]. To escape generic scanning, a virus can modify its code and alters its appearance on each infection. The techniques that have been employed to achieve this end range from *encryption* to *polymorphic* techniques, to modern *metamorphic* techniques [17].

2.1.1 Encrypted Viruses

The simplest way to change the appearance of a virus is to use encryption. An encrypted virus consists of a small decrypting module (a decryptor) and an encrypted virus body. If a different encryption key is used for each infection, the encrypted virus body will look different. Typically, the encryption method is rather simple, such as `xor` of the key with each byte of the virus body. Simple `xor` is very practical because `xor`ing the encrypted code with the key again will give the original code and so a virus can use the same routine for both encryption and decryption.

With encryption, the decryptor remains constant from generation to generation. As a result, detection is possible based on the code pattern of the decryptor. A scanner that cannot decrypt or detect the virus body directly can recognize the decryptor in most cases.

2.1.2 Polymorphic Viruses

To overcome the problem of encryption, namely the fact that the decryptor code is long and unique enough for detection, virus writers started implementing techniques to create mutated decryptors. Polymorphic viruses can change their decryptors in newer generations. They can generate a large number of unique decryptors which use different encryption method to encrypt the virus body. A polymorphic virus thus has no parts that stay constant on each infection.

To detect polymorphic viruses, anti-virus software incorporates a code emulator which dynamically decrypts the encrypted virus body. Because all polymorphic viruses carry a constant virus body, detection is still possible based on the decrypted virus code.

2.1.3 Metamorphic Viruses

To make viruses more robust to emulation, virus writers developed numerous advanced metamorphic techniques. According to Muttik [11], “Metamorphics are body-polymorphics”. A metamorphic virus not only changes it decryptor on each infection but

also its virus body. New virus generations look different from one another and they do not decrypt to a constant virus body. A metamorphic virus changes its "shape" but not its behavior. This is illustrated diagrammatically by Szor in [17], and is shown in Figure 1.

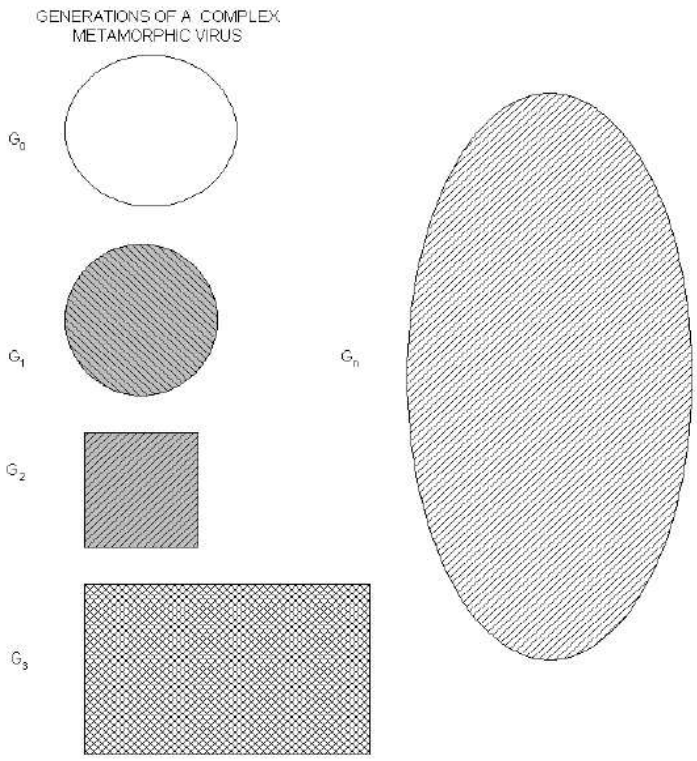


Figure 1 The multiple shapes of a metamorphic virus body, reproduced from [17].

Different techniques have been implemented by virus writers to create mutated virus bodies. One of the simplest techniques employs *register usage exchange*; an example is the W95/Regswap virus [16]. With this technique, a virus uses the same code but different registers in a new generation. Such viruses can usually be detected by a wildcard string [16].

A stronger technique employs *permutation* to reorder a virus's subroutines, as seen in the W32/Ghost virus [16]. With *n* different subroutines, a virus can generate *n*! different virus generations. W32/Ghost has 10 subroutines and so it has 10! = 3,628,800 variations. Even with the high number of subroutine combinations, the virus may still be detected with search strings [16].

More complex metamorphic viruses *insert garbage instructions* between core instructions. Garbage instructions are instructions that are either not executed or have no effect on program outcomes [10]. An example of the former is the `nop` instruction while "`add eax, 0`" and "`sub ebx, 0`" are sample instructions that do not affect program results. Alternatively, metamorphic viruses *insert jump instructions* into their code to point to the next instruction of the virus code. The Win95/Zperm family of viruses creates new mutations by removal and insertion of jump and garbage instructions as illustrated in Figure 2 [16].

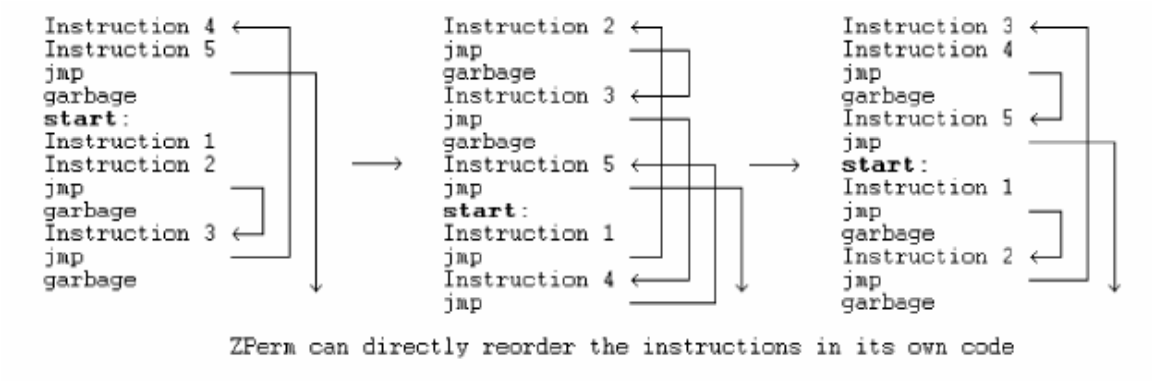


Figure 2 The Zperm virus, reproduced from [16].

Another common metamorphic technique is *substitution*, which is the replacement of an instruction or group of instructions with an equivalent instruction or group. For example, a conditional jump (Jcc) can be replaced by JNcc with inverted test condition and swapped branch labels [21]. A "push ebp; mov ebp, esp" sequence can be replaced by "push ebp; push esp; pop ebp" [16]. Sometimes, viruses implement *instruction opcode changes*. For example, to zero out the register eax, we can either xor its content with itself or use sub to achieve the same result. In other words, "xor eax, eax" can be replaced by "sub eax, eax" [16].

Transposition, or rearrangement of instruction order, is another technique used by metamorphic viruses. Instruction reordering is possible if no dependency exists between instructions. Consider the following example from [21]:

```
op1 [r1] [, r2]
op2 [r3] [, r4]                ; here r1 and/or r3 are to be modified
```

Swapping of the two instructions is allowed if

1) r1 not equal to r4; and
2) r2 not equal to r3; and
3) r1 not equal to r3.

Depending on the implemented techniques, a metamorphic virus can be very complex and very hard to detect even with present day detection techniques. Unlike polymorphic viruses, which decrypt themselves to a constant virus body in memory and provide a complete snapshot of the decrypted virus body during its execution, metamorphic viruses do not become constant anytime anywhere. The detection of metamorphic viruses has been and will likely to continue to be an active research area.

2.1.4 Virus Construction Kits

Viruses are mostly written in assembly language, and not too many people can manage to write complicated and functional assembly code. Some virus-writing groups try to make the virus creation process quick and easy. They make available many virus construction kits which can generate all kinds of malicious programs like viruses, worms, Trojan horses and logic bombs. Virtually any type of virus can be created – DOS COM / EXE viruses, 16-bit / 32-bit Windows viruses, script viruses, macro viruses, PE viruses, etc [16]. These toolkits are designed to be simple to use and some even come with commercial-grade interactive graphical interfaces. The tools allow anybody, novice or expert, to generate malicious code quickly and easily.

User-friendly as they are, some of these tools are also built with very sophisticated features such as anti-disassembly, anti-debugging, anti-emulation, and anti-behavior blocking. Some kits come equipped with code morphing ability which allows them to produce different-looking viruses. In this sense, the viruses they produce are metamorphic, not just polymorphic. The more popular ones among the 150+ generators available at the VX Heavens [19] include:

- PS-MPC (Phalcon/Skism Mass-Produced Code generator)
- G2 (Second Generation virus generator)
- MPCGEN (Mass Code Generator)
- NGVCK (Next Generation Virus Creation Kit)
- VCL32 (Virus Creation Lab for Win32)

2.2 Antivirus Defense Techniques

As computer viruses evolve and become more complex, antivirus software must become more sophisticated to defend against virus attacks. This section discusses the virus detection techniques that have been deployed over the years. These techniques include:

1) pattern-based scanning in first-generation scanners;
2) nearly exact and exact identification in second-generation scanners;
3) code emulation;
4) heuristic analysis to detect new and unknown viruses [16].

2.2.1 First Generation Scanners

The simplest approach to virus detection is string scanning. First generation scanners look for "virus signatures" which are sequences of bytes (strings) extracted from viruses in files or in memory. A good signature for a virus consists of sequences of text strings or byte codes found commonly in the virus but infrequently in benign programs. Usually, a human expert converts the virus binary code into assembly code, looks for sections that signify viral activities and picks the corresponding bytes in the machine code to be the virus signature. More efficient methods use statistical techniques to extract good signatures automatically [5].

Virus signatures are organized into databases. To identify virus infection, virus scanners check specific areas in files or system areas and match them against known signatures in databases. Some simple scanners also support wildcard search strings, such as "??02 33C9 8BD1 419C" where the wildcard is indicated by '?'. Wildcard strings allow skipped bytes and regular expressions and can sometimes be used to detect encrypted or even polymorphic viruses [16]. Using a search string from the common code areas of all known variants of a virus to scan for the virus family is known as generic detection [16]. A generic string typically contains wildcards.

To speed up detection, some scanners search only the start and the end of a file instead of the entire file as early computer viruses are mostly prepending (i.e., attached to the front

of the host programs) or appending (i.e., attached to the end of the hosts). Faster scanners look for entry-points, which are common targets of computer viruses, in the headers of executable files.

2.2.2 Second Generation Scanners

Second-generation scanners refine the detection process to detect viruses that evolve to mutate their body. Smart scanning ignores junk instructions like nop and excludes them in virus signatures. Nearly exact identification uses double strings, cryptographic checksums, or hash functions to achieve higher speed and greater accuracy. Exact identification uses all (as opposed to one in nearly exact identification) constant ranges of the virus bytes to calculate a checksum. Exact identification scanners are usually slower than simple scanners but a well-written one can differentiate virus variants precisely.

2.2.3 Code Emulation

With code emulation, anti-virus software implements a virtual machine to simulate CPU and memory activities. Scanners execute the virus code on the virtual machine rather than on the real processor. Depending on how well the virtual machine mimics system functionalities, few viruses are able to recognize that they are confined and examined in a virtual environment.

Code emulation is a very powerful technique, particularly in dealing with encrypted and polymorphic viruses. Encrypted and polymorphic viruses decrypt themselves in memory.

If an emulator is run long enough, the decrypted virus body will eventually present itself to a scanner for detection. The scanner can check its virtual machine's memory when a maximum number of iterations or other stop conditions are met. Alternatively, string scanning can be done periodically every predefined number of iterations. In this way, complete decryption of the virus body is not necessary as long as the decrypted part is long enough for identification. Code emulation can also be applied to metamorphic viruses that use single or multiple encryptions.

Code emulation can become too slow to be useful if the decryption loop is very long, particularly when a virus inserts garbage instructions in its polymorphic decryptor. A new decryption technique uses code optimization to reduce the polymorphic decryptor to its core instruction set. As the emulator iterates through the decryption loop, it removes junk and other instructions that do not change program state. Code optimization speeds up emulation and provides a profile of the decryptor for detection [16].

2.2.4 Heuristic Analysis

Heuristic analysis is used to detect new or unknown viruses. Often times, it is used to detect variants of an existing virus family. Heuristic methods can be static or dynamic. Static heuristics base the analysis on file format and the code structure of virus fragments. Dynamic heuristics use code emulation to simulate the processor and operating system and detect suspicious operations while the virus code is executed on a virtual machine.

Heuristic analysis is prone to false positives. A false positive occurs when a heuristic analyzer incorrectly tags a benign program as viral. These false alarms are not cost-effective. Too many false positives destroy users' trust and make a system more vulnerable as users may mistakenly assume a false alarm when it is a real attack.

2.3 Use of Machine Learning Techniques

Various researchers have attempted to use machine learning techniques to perform heuristic analysis on metamorphic viruses. This section covers the result and potential of some of the techniques, which include:

1) data mining methods;
2) use of neural networks;
3) use of hidden Markov models.

2.3.1 Data Mining Approach

Data mining methods are often used to detect patterns in a large set of data. These patterns are then used to identify future instances in a similar type of data. Schultz et al. experimented with a number of data mining techniques to identify new malicious binaries [14]. They used three learning algorithms to train a set of classifiers on some publicly-available malicious and benign executables. They compared their algorithms to a traditional signature-based method and reported a higher detection rate for each of their algorithms. However, their algorithms also resulted in higher false positive rates when compared to signature-based method.

The key to any data mining framework is the extraction of features, which are properties extracted from examples in the dataset. Schultz et al. extracted some static properties of the binaries as features. These include system resource information (the list of DLLs, the list of DLL function calls, and the number of different function calls within each DLL) obtained from the program header, and consecutive printable characters found in the files. The most informative feature they used was byte sequences, which were short sequences of machine code instructions generated by the hexdump tool.

The features were used in three different training algorithms. There was an inductive rule-based learner that generated Boolean rules to learn what a malicious executable was; a probabilistic method that applied Bayes rule to compute the likelihood of a particular program being malicious, given its set of features; and a multi-classifier system that combined the output of other classifiers to give the most likely prediction.

2.3.2 Using Neural Networks

Researchers at IBM implemented a neural network for heuristic detection of boot sector viruses [18]. The features they used were short byte strings, called trigrams, which appear frequently in viral boot sectors but not in clean boot sectors. They extracted about 50 features from a corpus of training data, which consisted of both viral and legitimate boot sectors. Each sample in the dataset was then represented by a Boolean vector indicating the presence or absence of these features.

The network was single-layered with no hidden units. It was trained using classic backpropagation technique. One common problem with neural network is overfitting, which occurs when a network is trained to identify the training set but fails to generalize to unseen instances. To eliminate this problem, multiple networks were trained using different features and a voting scheme was used to determine the final prediction.

The neural network was able to identify 80-85% of viral boot sectors in the validation set with a false positive rate of less than 1%. The neural network classifier has been incorporated into the IBM AntiVirus software which has identified about 75% of new boot sector viruses since it was released [18]. A similar technique was later applied by Arnold and Tesauro to successfully detect Win32 viruses [1]. From [18], we can conclude that neural networks are very effective in detecting viruses closely related to those in the training set. They can also identify new families of viruses containing similar features as the training samples.

2.3.3 Using Hidden Markov Models

Hidden Markov models (HMMs) are well suited for statistical pattern analysis. Since their initial application to speech recognition problems in the early 1970's [12], HMMs have been applied to many other areas including biological sequence analysis [7].

An HMM is a state machine where the transitions between states have fixed probabilities. Each state in an HMM is associated with a probability distribution for observing a set of observation symbols. We can "train" an HMM to represent a set of data, which is usually in the form of observation sequences. The states in the trained HMM then represent the features of the input data, while the transition and the observation probabilities represent the statistical properties of these features. Given any observation sequence, we can match it against a trained HMM to determine the probability of seeing such a sequence. The probability will be high if the sequence is "similar" to the training sequences.

In protein modeling, HMMs are used to model a given family of proteins [8]. The states correspond to the sequence of positions in space while the observations correspond to the probability distribution of the 20 amino acids that can occur in each position. A model for a protein family assigns high probabilities to sequences belonging to that family. A trained HMM can then be used to discriminate family members from non-members.

Metamorphic viruses form families of viruses. Even though members in the same family mutate and change their appearances, some similarities must exist for the variants to maintain the same functionality. Detecting virus variants thus reduces to finding ways to detect these similarities. Hidden Markov models provide a means to describe sequence variations statistically. We propose to use HMMs similar to those used in protein sequence analysis to model virus families. In virus modeling, the states correspond to the features of the virus code, while the observations are instructions or opcodes making up

the program. A trained model should then be able to assign high probabilities to and thus identify viruses belonging to the same family as the viruses in the training set.

3. SIMILARITIES BETWEEN VARIANTS OF METAMORPHIC VIRUSES

It has generally been agreed that for a virus to escape detection, metamorphism is the best approach. Different generations of a virus must look different to avoid detection by signature-based scanning. Some of the virus creation toolkits that we mentioned in Section 2.1.4, including G2 (Second Generation virus generator) and NGVCK (Next Generation Virus Creation Kit), come with the ability to generate morphed versions of the same virus, even from identical configurations. In this section, we take a look at how "effective" these generators are, or how "different" are the variants generated by the same engine. We use a similarity index and also a graphically representation to display the similarity between two assembly programs.

3.1 Method to Compare Two Pieces of Code

To compare two pieces of code, we employed the method developed by Mishra in [9]. His method compares two assembly programs and assigns a quantitative score to represent the percentage of similarity between the two programs.

Mishra's method is outlined below and is illustrated graphically in Figure 3.

1) Given two assembly programs X, and Y for which we want to measure their similarity, we extract the sequence of opcodes for each of the programs, excluding comments, blank lines, labels, and other directives. The result is two opcode sequences of length n, and m, where n and m are the numbers of opcodes in programs X and Y, respectively.
2) We compare the two opcode sequences by considering all subsequences of three consecutive opcodes from each sequence. We count as a match any case where all three opcodes are the same in any order, and we mark on a graph the coordinate (x, y) of the match where x is the opcode number of the first opcode of the three-opcode subsequence in program X and y is the opcode number of the opcode subsequence in program Y.
3) After comparing the entire opcode sequences and marking all the match coordinates, we obtain a graph plotted on a grid of dimension $n \times m$. Opcode numbers of program X are represented on the x-axis and those of program Y are represented on the y-axis. To remove noise and random matches, we only graph those line segments of length greater than the threshold value five.
4) Since we are performing a sequential match between the two opcode sequences, identical segments of opcodes will form line segments of 45 degrees to either axis (i.e., having a slope of 1) on the graph. If a line falls right on the diagonal, the matching opcodes are at identical locations on the two opcode sequences. A line off the diagonal indicates that the matching opcodes appear at different locations in the two files.

5) For each axis, we count the number of opcodes that are covered by one or more of the 45 degree "match" line segments. This number is divided by the respective total number of opcodes (n for program X and m for program Y) to give the percentage of opcodes that match some opcodes in the other program. The similarity score for the two programs is the average of these two percentages.

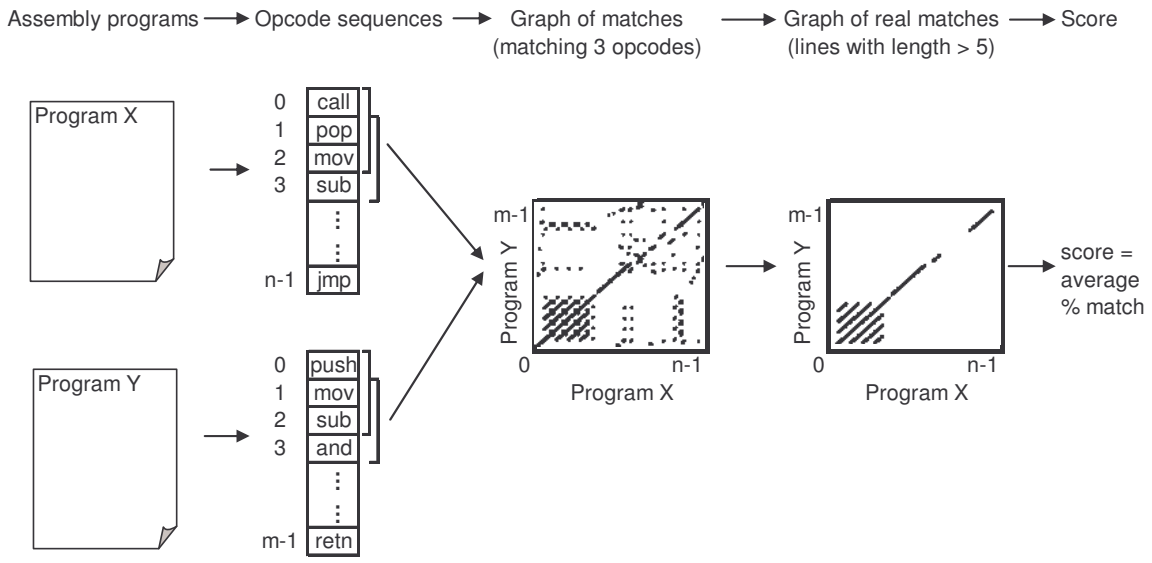


Figure 3 The process of finding the similarity between two assembly programs.

3.2 Test Data

We analyzed 45 viruses generated by four virus generators that we downloaded from VX Heavens [19]. We also compared some randomly chosen utility programs from the Cygwin DLL [2] to see how viruses differ from "normal" executable files. The programs that we analyzed include:

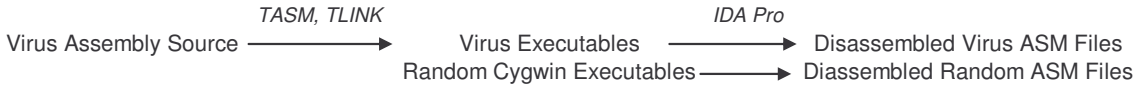
- 20 viruses generated by NGVCK (Next Generation Virus Creation Kit) version 0.30 released in June 2001;
- 10 viruses generated by G2 (Second Generation virus generator) version 0.70a released in January 1993;
- 10 viruses generated by VCL32 (Virus Creation Lab for Win32) released in February 2004;
- 5 viruses generated by MPCGEN (Mass Code Generator) version 1.0 released in 1993;
- 20 randomly chosen utility executables from the Cygwin DLL version 1.5.19.

The virus variants were named after their generators as follows:

- the 20 viruses generated by NGVCK were named NGVCK0 to NGVCK19;
- the 10 generated by G2 were named G0 to G9;
- the 10 generated by VCL32 were named VCL0 to VCL9;
- the 5 generated by MPCGEN were named MPC0 to MPC4.

The 20 random utilities files were named R0 to R19.

The viruses created by the virus generators were in assembly source code. To make virus executable files, we assembled them with the Borland Turbo Assembler TASM 5.0. The generated executables were then disassembled by the IDA Pro Disassembler [3] version 4.6.0. All the disassembling used the same default settings. The cygwin utilities were also disassembled by IDA Pro. The sequence of process is summarized as:



We added the prefix "IDA_" to the respective file names to denote that the files were disassembled ASM files created by IDA Pro and to distinguish them from the original ASM files. For example, the file disassembled from R0.EXE was named IDA_R0.ASM.

We compared the disassembled assembly (ASM) files instead of the original assembly codes generated by the virus generators. We believed by assembling and disassembling with the same tools using the same settings, we can eliminate some differences due to different coding style of the different virus writers. The standardized disassembling process makes for more accurate comparison when we compare the viruses generated by different generators, or when we compare viruses with random "normal" programs. It makes the similarity measure better reflect the effectiveness of the metamorphism employed. The process also simulates a more realistic scenario because when detecting viruses in real environment, what we have available are virus executables. That is, disassembling and analyzing the resultant assembly files is what we need to do in practice.

3.3 Test Result

For each of the virus generator, we compared each of the viruses it generated to all the other viruses generated by the same generator, to see how "effective" the generator is in

terms of generating different-looking virus variants. For each pair of virus variants under comparison, we computed their similarity score using the method described above in Section 3.1. Comparisons were also made between the random normal files. The raw similarity scores of all the comparisons are given in Table A-1 to Table A-5 in Appendix A. Figure 4 below is a scatter plot showing the similarity scores of the 190 comparisons among the 20 NGVCK viruses and the 190 comparisons among the 20 random files. Clearly, similarities between NGVCK virus variants are lower than those between random files.

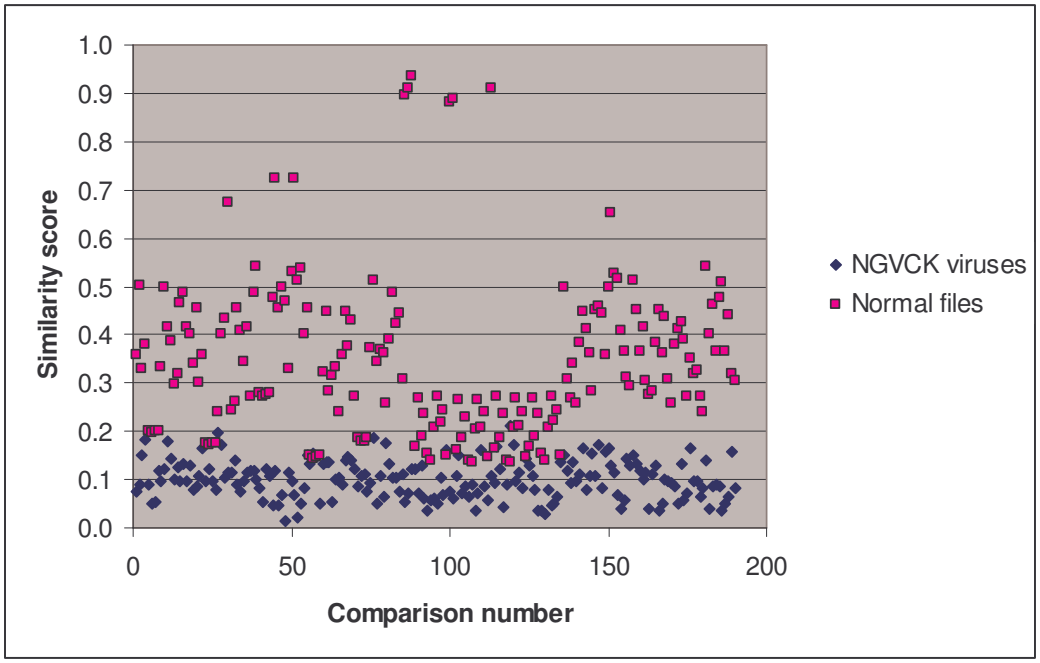


Figure 4 Scatter plot showing similarity scores between NGVCK virus variants and those between random "normal" files.

The minimum, maximum, and average scores of each generator and the normal files are summarized below in Table 1.

Minimum, maximum, and average similarity scores					
	NGVCK	G2	VCL32	MPCGEN	Normal
min	**0.01493**	0.62845	0.34376	0.44964	0.13603
max	**0.21018**	0.84864	0.92907	0.96568	0.93395
average	**0.10087**	0.74491	0.60631	0.62704	0.34689

Table 1 Minimum, maximum, and average similarity scores between virus variants generated by the generators and between random "normal" files.

Comparing the four generators, NGVCK generates viruses of the lowest similarities, which range from 1.5% to 21.0% with an average of about 10.0%. The other generators are not as effective at generating different-looking viruses. The similarities between two variants of the same virus range from 34.4% to 96.6%, and the average scores of G2, VCL32, and MPCGEN are 74.5%, 60.6%, and 62.7%, respectively. Compare to random normal files, which have an average similarity of 34.7%, we can see that the viruses that NGVCK generates are substantially different from one another, while the virus variants generated by the other generators are more similar to one another than random files.

These comparison results are represented graphically by the bubble graph in Figure 5. Here the minimum score is shown along the x-axis; the maximum score is shown along the y-axis; and the size of the bubble represents the average similarity. Under this representation, an effective generator would have a bubble that is very close to the origin

and also has a very small size, since effectively morphed variants of a virus should have low minimum, low maximum and low average similarities.

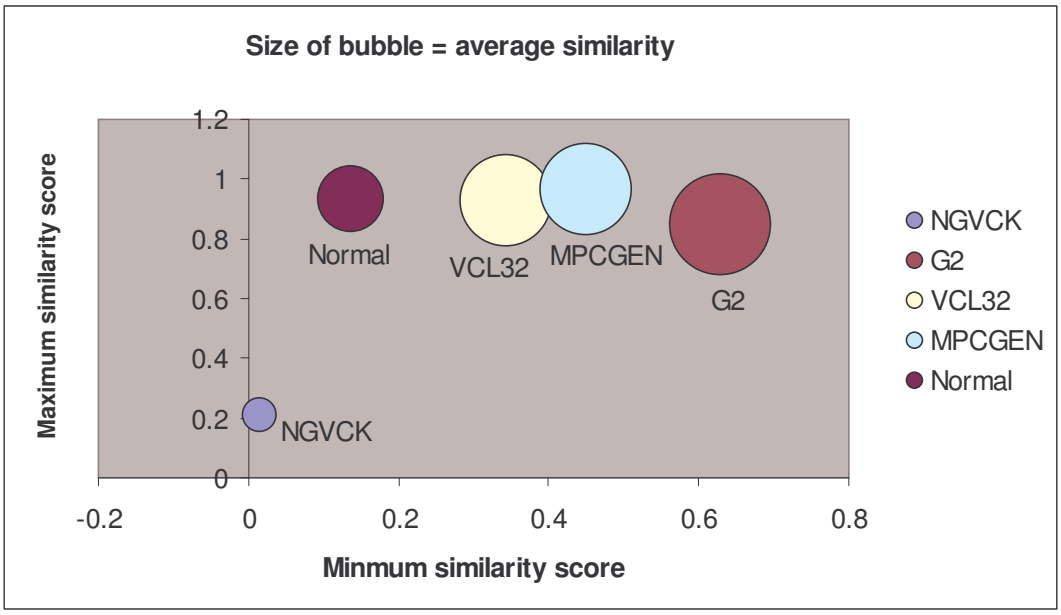


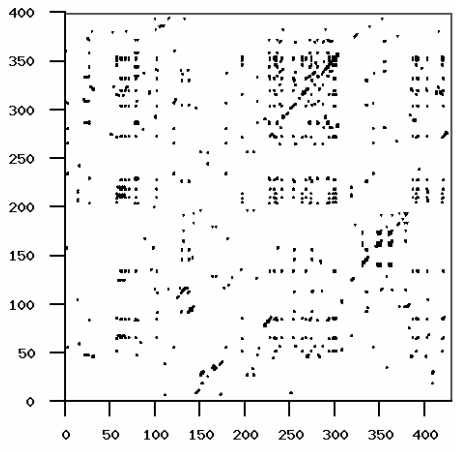
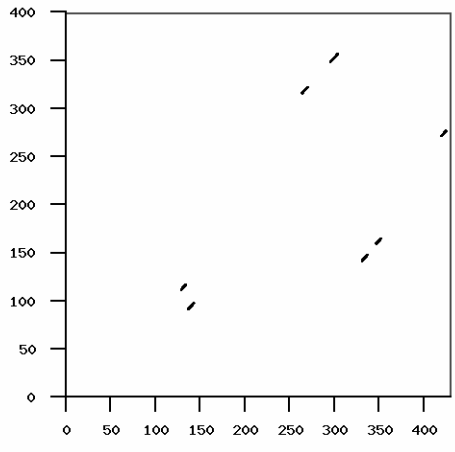
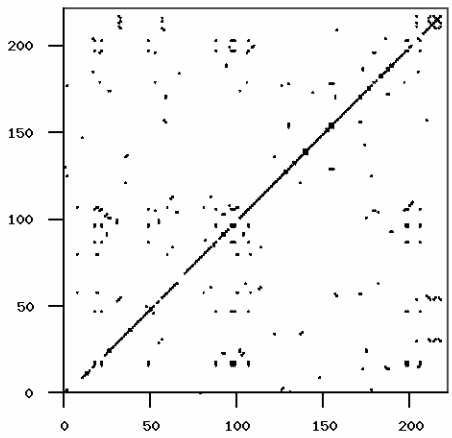
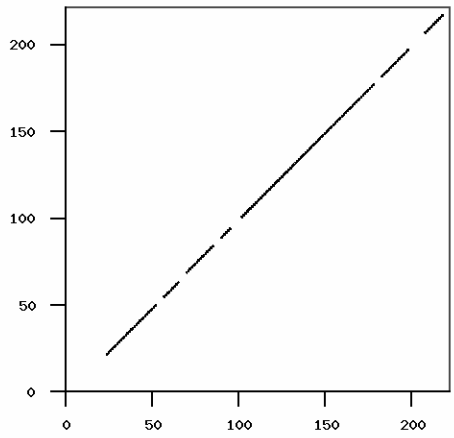
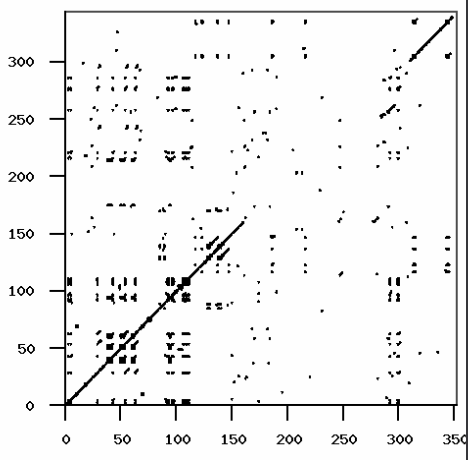
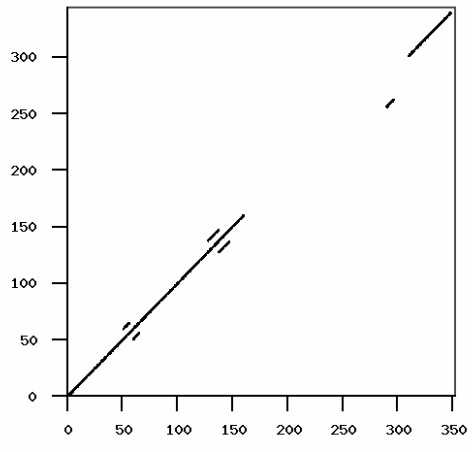
Figure 5 Bubble graph showing the minimum, maximum, and average similarity between virus variants generated by each of the generators and between random "normal" files.

As is shown in the graph, NGVCK clearly outperforms the other generators in terms of generating different-looking viruses. VCL32 and MPCGEN have similar morphing ability as their variants have comparable minimum, maximum, and average similarities. G2-viruses have a higher average similarity, as is represented by the bigger bubble size, although the maximum similarity of the variants is lower than that of VCL32- and MPCGEN-viruses. Random files have similarities higher than NGVCK-viruses but lower than virus variants produced by generators G2, VCL32, and MPCGEN.

The following table shows the similarity graphs of some of the virus pairs. For each generator, we chose a representative pair which has a similarity score close to the average similarity score, to illustrate how a typical virus pair differ from each other. The first column gives the virus names with their similarity score in parenthesis. The second column shows the graphs of all matches, as defined in Section 3.1 above. The third column shows the graphs of real matches after noise and random matches have been removed. The pairs selected and their scores are:

- IDA_NGVCK0 against IDA_NGVCK8, similarity = 11.9%
- IDA_G4 against IDA_G7, similarity = 75.2%
- IDA_VCL0 against IDA_VCL9, similarity = 60.2%
- IDA_MPC1 against IDA_MPC3, similarity = 58.0%
- random files IDA_R0 and IDA_R1, similarity = 35.7%.

Virus Pair (Similarity score)	Graph of all matches (matching 3 consecutive opcodes in any order)	Graph of real matches (match of length > 5)

IDA_ NGVCK0- IDA_ NGVCK8 (11.9%)		
IDA_G4- IDA_G7 (75.2%)		
IDA_VCL 0- IDA_VCL 9 (60.2%)		

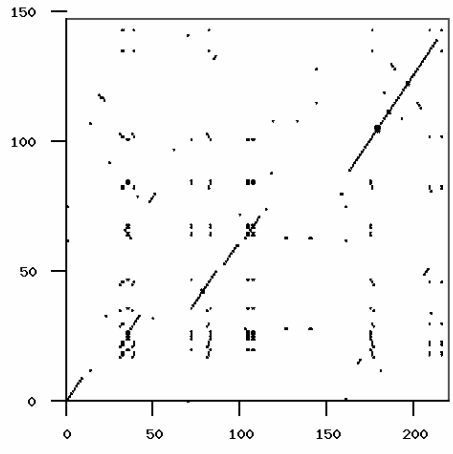
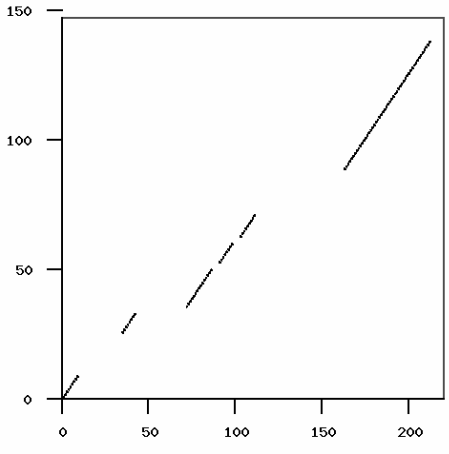
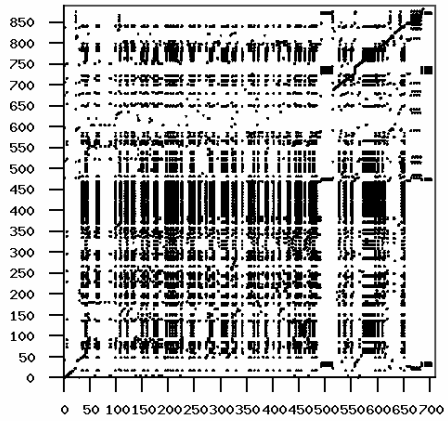
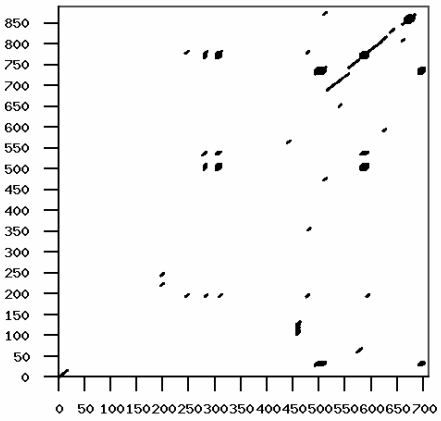
IDA_MPC 1- IDA_MPC 3 (58.0%)		
IDA_R0- IDA_R1 (35.7%)		

Table 2 Similarity graphs of 4 selected virus pairs and one random file pair.

If we take a closer look at the graphs for the pair of G2-viruses and the pair of VCL32-viruses, we can see that the real matches are almost all along the diagonal. This indicates that virus variants of the same virus have identical opcodes at identical positions. This is obviously not very effective metamorphism. On the other hand, the matches between the MPCGEN-virus pair are off the diagonal, which shows that identical opcodes appear in different positions of the two virus variants. From this evidence, we can say that

MPCGEN has a greater morphing ability than the other two generators. NGVCK is the most effective in the sense that the match segments are very short and that they are way off the diagonal. Even if we look at the pair that has the highest similarity (IDA_NGVCK7 and IDA_NGVCK14, similarity = 21.0%), the match segments are still short and off the diagonal. The two similarity graphs of this pair are shown below.

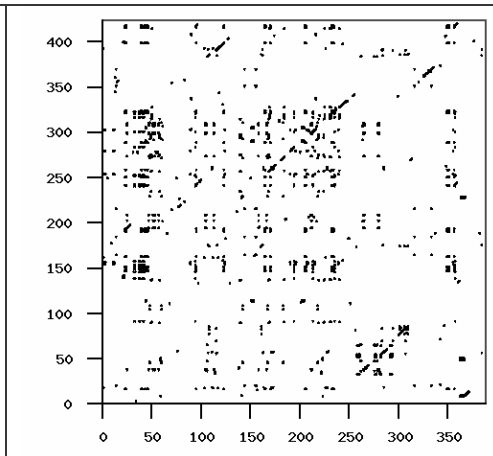
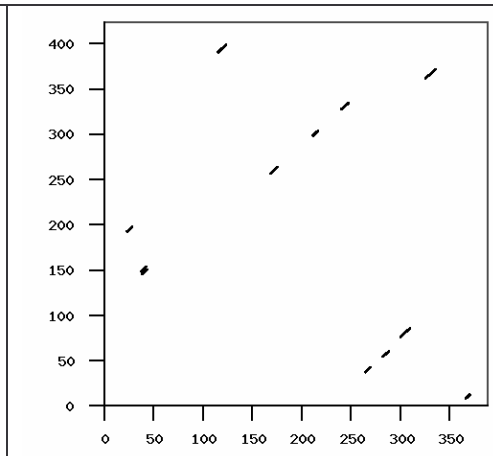
IDA_ NGVCK7- IDA_ NGVCK14 (21.0%)		

Table 3 Similarity graphs of the NGVCK-virus pair that has the highest similarity.

4. HIDDEN MARKOV MODELS TO DETECT VIRUSES IN SAME FAMILY

In this project, we developed a system to train multiple hidden Markov models (HMMs) on a set of metamorphic virus variants. The trained models were tested for their ability to detect morphed variants of the same virus. The effectiveness of the HMM approach is determined by the detection rate, the number of false positives and false negatives, and the overall accuracy.

4.1 Theory and Algorithms for Hidden Markov Models

A hidden Markov model is a statistical model that describes a series of observations generated by a stochastic process, or Markov process. A Markov process is a sequence of states, where the progression to the next state depends solely on the present state but not on the past states. The Markov process in an HMM is "hidden"; what we can see is the sequence of observations associated with the states. Our goal is to make use of the observable information to gain insight into various aspects of the underlying Markov process [15].

We illustrate these concepts by an example taken from [15]. Suppose we want to know the average annual temperature of a particular location over a preceding period of several consecutive years and suppose that there is no recording of past temperature of any form for this location. Since there is no way to know the year-to-year temperature directly, we look for evidence to predict the temperature indirectly.

For simplicity, we consider only two possible annual temperatures: "hot" (*H*) or "cold" (*C*). Suppose we know that the probability of a hot year followed by another hot year is 0.7 and that of a cold year followed by another cold year is 0.6. This information can be represented by the matrix:

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}.$$

Now assume research result tells us that the tree ring size of a certain kind of tree, whether it is small (*S*), medium (*M*), or large (*L*), is related to the annual temperature as:

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

meaning that in a hot year, the probability of a tree having a small, medium, or a large tree ring is 0.1, 0.4 and 0.5 respectively. If we observe the tree ring sizes for such a tree, we can use this information to deduce the possible annual temperatures over the years of interest.

In this example, the temperatures (*H* and *C*) are the states and the transition of temperature from year to year defines the Markov process. Tree ring sizes (*S*, *M*, *L*) are the observable outcomes and the probabilities of seeing the different tree ring sizes at each temperature represent the probability distribution of the observation symbols at each state. The actual states are "hidden" since we cannot directly observe the temperatures. What we can see are the observations (tree ring sizes) and these are related to the states statistically.

Suppose we represent the observation symbols *S*, *M*, *L* by 0, 1, 2 respectively and suppose that a particular four-year series of observed tree ring sizes is given by the observation sequence $O = (0, 1, 0, 2)$. We might want to find the most likely state sequence of the Markov process that generates the observation sequence. In other words,

we may want to determine the most likely annual temperatures (H or C) over this series of four years from our observation of the tree ring sizes.

4.1.1 Notation

Let

T = the length of the observed sequence

N = the number of states in the model

M = the number of distinct observation symbols

O = the observation sequence = $\{O_0, O_1, \ldots, O_{T\text{-}1}\}$

Q = the set of states of the Markov process = $\{q_0, q_1, \ldots, q_{N\text{-}1}\}$

V = the set of observation symbols = $\{0, 1, \ldots M - 1\}$

A = the state transition probability distributions

B = the observation probability distributions

π = the initial state distribution

$\lambda = (A, B, \pi)$ = the HMM defined by its parameter A, B, and π.

Figure 6 shows a generic HMM. The state and observation at time t are represented by X_t and O_t respectively. The Markov process, which is hidden behind the dashed line, is determined by the initial state X_0 and the A matrix. What we can observe are the observations O_t, which are related to the states of the Markov process by the B matrix.

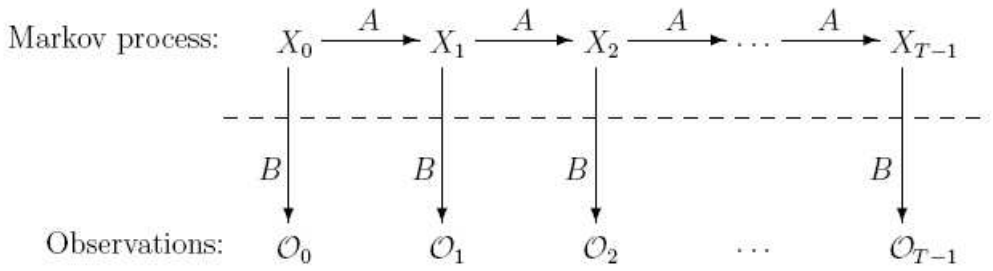


Figure 6 A generic hidden Markov model, reproduced from [15].

For our temperature example, the state transition matrix A is defined by the probabilities of temperature transitions from year to year; the observation matrix B is defined by the probabilities of observing the tree ring sizes. That is,

$$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}, \text{ and}$$

$$B = \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix}$$

which are the same matrices given previously.

The matrix $A = \{a_{ij}\}$ is $N \times N$ with

$$a_{ij} = P(q_j \text{ at } t+1 \mid q_i \text{ at } t)$$

representing the probability of making a transition from state q_i at time t to state q_j at time $t+1$.

The matrix $B = \{bj(k)\}$ is $N \times M$ with

$$b_j(k) = P(\text{observation } k \text{ at } t \mid \text{state } q_j \text{ at } t)$$

representing the probability of observing symbol k at time t given we are in state q_j at time t.

The matrix $\pi = \{\pi_i\}$ is $1 \times M$ with

$$\pi_i = P(q_i \text{ at } t = 0)$$

representing the probability of being initially in state q_i at time 0. We assume for the temperature example that $\pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$.

The matrices A, B, and π make up the parameters of an HMM. Note that A, B, π are row stochastic, i.e., each row of these matrices represents a probability distribution and therefore must sum to 1 [15].

For a generic state sequence $X = (x_0, x_1, x_2, x_3)$ of length four, with corresponding observations $O = (O_0, O_1, O_2, O_3)$. The probability of the state sequence X is given by

$$P(X \mid \lambda) = \pi_{x0}\, b_{x0}(O_0)\, a_{x0,\,x1}\, b_{x1}(O_1)\, a_{x1,\,x2}\, b_{x2}(O_2)\, a_{x2,\,x3}\, b_{x3}(O_3)$$

where π_{x0} is the probability of starting in state x_0, $b_{x0}(O_0)$ is the probability of observing O_0 at x_0 and $a_{x0,\,x1}$ is the probability of transiting from state x_0 to state x_1. This easily generalizes to a sequence of any length.

In our temperature example, with observation sequence $O = (0, 1, 0, 2)$, we can compute the probability of this observation sequence having been generated by each four-state

sequence. For example, the probability that observation O was generated by the state sequence $HHCC$ is

$$P(HHCC) = 0.6(0.1)(0.7)(0.4)(0.3)(0.7)(0.6)(0.1) = 0.000212$$

In the same manner, we can compute the probability of each of the possible state sequences of length four, given the fixed observation sequence O. These probabilities are listed in Table 4. We will have some more to say about these probabilities when we discuss the HMM algorithms.

state sequence	probability
HHHH	0.000412
HHHC	0.000035
HHCH	0.000706
HHCC	0.000212
HCHH	0.000050
HCHC	0.000004
HCCH	0.000302
HCCC	0.000091
CHHH	0.001098
CHHC	0.000094
CHCH	0.001882
CHCC	0.000564
CCHH	0.000470
CCHC	0.000040
CCCH	0.002822
CCCC	0.000847
Σ probability	0.009629
max probability	0.002822

Table 4 Probabilities of observing O = (0, 1, 0, 2) for all possible 4-state sequences.

In general, the three problems that we are interested in solving with an HMM are [15]:

- Given the model $\lambda = (A, B, \pi)$ and an observation sequence O, find $P(O \mid \lambda)$. That is, find the likelihood of observing the sequence O given the model.
- Given $\lambda = (A, B, \pi)$ and an observation sequence O, find an optimal state sequence that could have generated O. (This is what we wanted to do in the temperature example above.) Note that "optimal" here has at least two interpretations. We can reasonably define optimal as:
    1) the state sequence with the highest probability from among all possible state sequences; or
    2) the state sequence that maximizes the expected number of correct states.
- Given an observation sequence O, the number of states N, and the number of symbols M, find the model parameters, i.e., the probabilities in the A, B, and π matrices, that maximize the probability of observing O. This is a discrete hill climb on the (A, B, π)-parameter space. In other words, we re-adjust the model parameters to best fit the observations.

4.1.2 Algorithms

There exist efficient algorithms to solve the three problems listed above. A thorough review of these algorithms can be found in [12] and [4]. In this section, we look at some of these algorithms, which include:

- the *Forward-Backward* algorithm for calculating the probability of being in a state q_i at time t given an observation sequence O;
- the *Viterbi* algorithm for finding the most likely state sequence given O; and
- the *Baum-Welch* algorithm for iteratively re-estimating the parameters A, B, π.

4.1.2.1 Finding the likelihood of an observation sequence O: the Forward algorithm

In the previous section, we saw that the probability of an observation sequence $O = (O_0, O_1, \ldots, O_{T\text{-}1})$ generated by a particular state sequence $X = (x_0, x_1, \ldots, x_{T\text{-}1})$ given a model λ is given by

$$P(O, X \mid \lambda) = \pi_{x0}\, b_{x0}(O_0)\, a_{x0,\,x1}\, b_{x1}(O_1)\, a_{x1,\,x2} \ \ldots \ a_{xT\text{-}2,\,xT\text{-}1}\, b_{xT\text{-}1}(O_{T\text{-}1}).$$

To find the probability of observing the sequence O, we generate all possible state sequences X_i of length T and sum over the probabilities $P(O, X_i \mid \lambda)$.

$$P(O \mid \lambda) = \sum_{X_i} P(O, X_i \mid \lambda)$$

$$= \sum_{X_i} \pi_{x_0} b_{x_0}(O_0) a_{x_0,x_1} b_{x_1}(O_1) a_{x_1,x_2} \ldots a_{x_{T-2},x_{T-1}} b_{x_{T-1}}(O_{T-1})$$

Going back to our temperature example, the probability of observing tree ring sizes $O = (0, 1, 0, 2)$ given our model is equal to the sum of all the probabilities listed in Table 1, which is 0.009629.

The probability $P(O \mid \lambda)$ tells us how well the observation sequence O matches the HMM λ. If λ has N states and O has length T, then there are N^T possible state sequences. Finding the probability $P(O, X_i \mid \lambda)$ for one of the state sequence X_i requires about $2T$ multiplications and so a direct computation of the summation requires about $2TN^T$ computations, which is infeasible even for small HMMs.

Instead of generating all possible state sequences, we use the Forward algorithm (sometimes called the α-pass) to compute this probability efficiently. For $t = 0, 1, \ldots, T - 1$ and $i = 0, 1, \ldots, N - 1$, define a forward variable

$$\alpha_t(i) = P(O_0, O_1, ..., O_t, x_t = q_i \mid \lambda)$$

which denotes the probability of observing the partial sequence $(O_0, O_1, \ldots, O_t)$ up to time t and being in state q_i at time t.

The forward variables can be found recursively using the following recurrence relation:

Step 1 Initialization:

$$\alpha_0(i) = \pi_i \, b_i(O_0), \qquad \text{for } i = 0, 1, \ldots, N - 1$$

Step 2 Induction:

$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} \right] b_i(O_t), \qquad \text{for } t = 1, 2, \ldots, T - 1 \text{ and } i = 0, 1, \ldots, N - 1.$$

Figure 7 illustrates the inductive process of finding $\alpha_t(i)$ using the variables $\alpha_{t-1}(j)$.

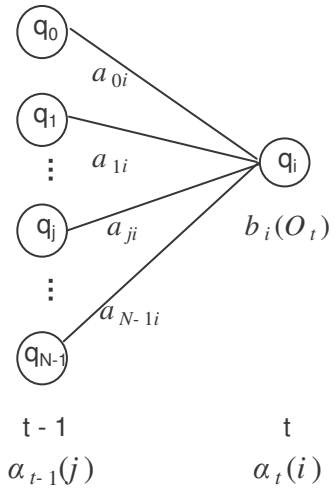


Figure 7 The inductive process of finding $\alpha_t(i)$ from variables $\alpha_{t-1}(j)$.

The probability of observing the sequence O given the model λ, $P(O \mid \lambda)$, can then be calculated as

$$P(O \mid \lambda) = \sum_{i=0}^{N-1} P(O_0, O_1, ..., O_T, x_T = q_i \mid \lambda)$$

$$= \sum_{i=0}^{N-1} \alpha_{T-1}(i) .$$

The recursive computation requires N^2T multiplications, which is much better than $2TN^T$ for the naive approach.

4.1.2.2 Finding the most likely state sequence: the Viterbi algorithm

Given an observation sequence $O = (O_0, O_1, \ldots, O_{T-1})$ and an HMM λ, the Viterbi algorithm finds a highest scoring overall path X^* that maximizes the probability $P(O, X \mid \lambda)$. We can determine the state sequence that is mostly likely to occur given the observation sequence.

For $t = 0, 1, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$, let $\delta_t(i)$ denotes the probability of the most probable state path $(x_0, x_1, \ldots, x_t)$ that generates the partial sequence $(O_0, O_1, \ldots, O_t)$ up to time t and ending in state q_i,

$$\delta_t(i) = \max_{x_0 \ldots x_{t-1}} P(O_0, O_1, \ldots, O_{t,}, x_0, x_1, \ldots, x_{t-1,} x_t = q_i \mid \lambda)$$

The $\delta_t(i)$ values can be found recursively as follows:

Step 1 Initialization:

$$\delta_0(i) = \pi_i \, b_i(O_0), \qquad \text{for } i = 0, 1, \ldots, N-1$$

Step 2 Induction:

$$\delta_t(i) = \max_{0 \le j \le N-1} \left[\delta_{t-1}(j) a_{ji} \right] b_i(O_t), \qquad \text{for } t = 1, 2, \ldots, T-1 \text{ and } i = 0, 1, \ldots, N-1.$$

At each successive t, the algorithm gives the probability of the best path ending at each of the states $i = 0, 1, \ldots, N-1$. Consequently, the probability of the most likely state sequence for the observation sequence O is

$$P^* = \max_{0 \le i \le N-1} \left[\delta_{T-1}(i) \right]$$

The Viterbi algorithm is similar to the Forward algorithm, except that maximizations replace the summations in the recursive calculations. Notice that the $\delta_t(i)$ values are probabilities values only. To actually find the state sequence X^*, we can use back-pointers at each step to keep track of the best states chosen along the path. The path can then be extracted by backtracking from the highest-scoring final state.

For our temperature example given at the beginning of Section 4.1, the mostly likely state sequence is *CCCH*, having the highest probability of 0.002822 as shown in Table 1.

4.1.2.3 Finding the optimal model parameters: the Baum-Welch algorithm

One of the most useful features of an HMM is that we can efficiently re-adjust the model parameters to best fit the observations. Given the matrix dimensions N and M, we can iteratively re-estimate the elements of A, B, and π so that the probability of observing an observation sequence O is maximized.

Before we discuss the re-estimation algorithm, let us first take a look at the Backward algorithm, or β-pass, which is analogous to the α-pass given above.

For $t = 0, 1, \ldots, T - 1$ and $i = 0, 1, \ldots, N - 1$, define the backward variable

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_{T-1} \mid x_t = q_i, \lambda)$$

which denotes the probability of observing the partial sequence (O_{t+1}, O_{t+2}, …, O_{T-1}) given we are in state q_i at time t.

$\beta_t(i)$ measures the probability after time t and can be obtained recursively starting at the end of the sequence:

Step 1 Initialization:

$$\beta_{T-1}(i) = 1, \quad \text{for } i = 0, 1, \ldots, N - 1$$

Step 2 Induction:

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad \text{for } t = T - 2, T - 1, \ldots, 0 \text{ and } i = 0, 1, \ldots, N - 1.$$

Figure 8 illustrates the recursive process.

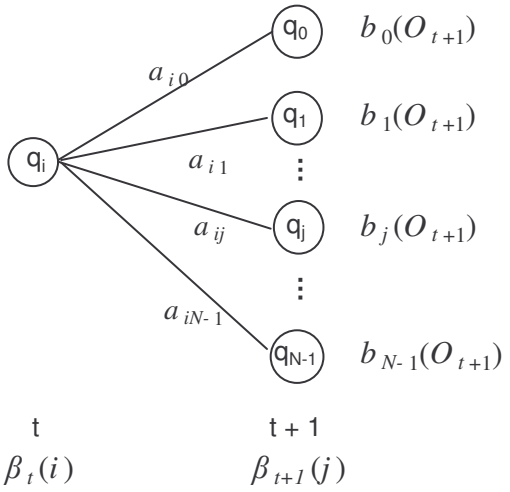


Figure 8 The inductive process of finding $\beta_t(i)$ from variables $\beta_{t+1}(j)$.

The Backward algorithm also gives us the probability of observing the sequence O given the model λ, or $P(O \mid \lambda)$, which should be the same number produced by the Forward algorithm:

$$P(O \mid \lambda) = \sum_{i=0}^{N-1} \pi_i b_i(O_0) \beta_0(i) .$$

Now, define the probability of being in state q_i at time t given the observation sequence O and the model λ, for $t = 0, 1, \ldots, T - 2$ and $i = 0, 1, \ldots, N - 1$, as

$$\gamma_t(i) = P(x_t = q_i \mid O, \lambda) .$$

This probability can be obtained from the forward-backward variables as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=0}^{N-1} \alpha_t(i)\beta_t(i)}$$

since $\alpha_t(i)$ accounts for the observations up to time t and $\beta_t(i)$ accounts for the observations after time t given we are in state q_i at time t. The denominator $P(O \mid \lambda) = \sum_{i=0}^{N-1} \alpha_t(i)\beta_t(i)$ is the normalization factor, which makes $\gamma_t(i)$ a probability distribution and sum to 1.

Next, define the joint probability of being in state q_i at time t and transiting to state q_j at time $t + 1$, for $t = 0, 1, \ldots, T - 2$ and $i, j \in \{0,1,..., N - 1\}$, as

$$\gamma_t(i, j) = P(x_t = q_i, x_{t+1} = q_j \mid O, \lambda).$$

This probability can be written in terms of α, β, A, and B as

$$\gamma_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O \mid \lambda)}.$$

The relationship among these probabilities is illustrated graphically in Figure 9.

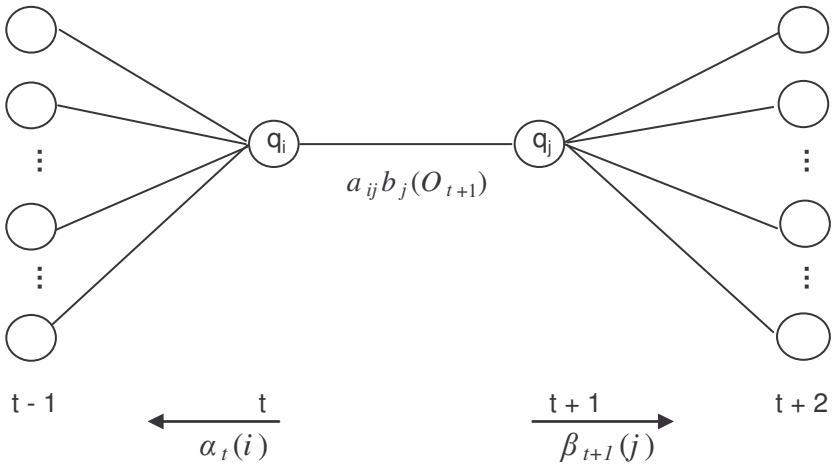


Figure 9 The variables for the computation of the joint probability $\gamma_t(i,j)$.

The $\gamma_t(i)$ and $\gamma_t(i, j)$ are related by

$$\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i, j)$$

$\gamma_t(i)$ gives us the probability of being in state q_i at time t. If we sum the probability over all possible T, we get the expected number of transitions from state q_i to any state. $\gamma_t(i, j)$ gives us the joint probability of being in state q_i at time t and in state q_j at time $t + 1$. The summation of $\gamma_t(i, j)$ over T thus gives the expected number of transitions from state q_i to state q_j. In other words,

$\sum_{t=0}^{T-2} \gamma_t(i)$ = the expected number of transitions from state q_i to any state, and

$\sum_{t=0}^{T-2} \gamma_t(i, j)$ = the expected number of transitions from state q_i to state q_j.

We can now re-estimate the parameters of $\lambda = (A, B, \pi)$ using the following formulae:

For $i = 0, 1, \ldots, N-1$,

$$\bar{\pi}_i = \gamma_0(i)$$

$$= \text{probability of being in state } q_i \text{ at } t = 0.$$

For $i = 0, 1, \ldots, N-1$ and $j = 0, 1, \ldots, N-1$,

$$\bar{a}_{ij} = \sum_{t=0}^{T-2} \gamma_t(i, j) \Big/ \sum_{t=0}^{T-2} \gamma_t(i)$$

$$= \frac{\text{Expected number of transitions from } q_i \text{ to } q_j}{\text{Expected number of transitions out of } q_i}$$

For $j = 0, 1, \ldots, N-1$ and $k = 0, 1, \ldots, M-1$,

$$\bar{b}_j(k) = \sum_{\substack{t=0 \\ O_t = k}}^{T-2} \gamma_t(j) \Big/ \sum_{t=0}^{T-2} \gamma_t(j)$$

$$= \frac{\text{Expected number of times the model is in state } q_j \text{ with observation } k}{\text{Expected number of times the model is in state } q_j}$$

We re-estimate λ iteratively until $P(O \mid \lambda)$ does not increase (or the increase is less than a predefined threshold) or until the maximum number of iterations is reached. The complete Baum-Welch expectation-maximization (EM) algorithm can be summarized as:

6) Initialize $\lambda = (A, B, \pi)$ with a best guess. If no prior information is available, choose random $\pi_i \approx 1/N$, $a_{ij} \approx 1/N$, and $b_j(k) \approx 1/M$.

7) Calculate $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i)$ and $\gamma_t(i, j)$.

8) Re-estimate the model $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\pi})$, and calculate $P(O \mid \overline{\lambda})$.

9) Stop if $P(O \mid \overline{\lambda}) - P(O \mid \lambda)$ is less than the predefined threshold or the maximum number of iterations is reached; otherwise set $\lambda = \overline{\lambda}$ and goto (2).

4.1.2.4 Posterior state probabilities

The Viterbi algorithm given in Section 4.1.2.2 finds the most probable state path through the model. But as we mentioned in Section 4.1.1, there is a second interpretation as to what constitutes an "optimal" state sequence. Instead of finding the highest scoring overall path, as is done by the Viterbi algorithm, we may want to find the most probable state for each specific observation O_t in the observation sequence $O = (O_0, O_1, \ldots, O_{T-1})$. More generally, we may want to find the probability that observation O_t is generated by state q_i given the sequence O, i.e., $P(x_t = q_i \mid O, \lambda)$. This is called the posterior probability of state q_i at time t.

This posterior probability is exactly the $\gamma_t(i)$ variable defined above in Section 4.1.2.3, which is given by

$$P(x_t = q_i \mid O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)}.$$

Hence, the optimal path that finds the most probable state for each position is obtained by finding, for each $t = 0, 1, \ldots, T - 1$, the state q_i for which $\gamma_t(i)$ is maximum.

This state sequence is not necessarily the same as the highest scoring sequence found by the Viterbi algorithm. We may be more interested in this sequence that maximizes all posterior probabilities when there are many different paths that have probabilities very close to the most probable one, or when we want to know only the state assignment at a particular point *t* rather than the complete path. It is possible that this state sequence may not be particularly likely as a path through the HMM. Sometimes it is not even a legitimate path when some of the transitions between states are not allowed.

4.1.3 Implementation Issue: Underflow and Scaling

The HMM computations discussed in Section 4.1.2 require repeated multiplications of the transition and observation probability values. One major challenge in the implementation is to deal with these small products which tend to zero exponentially as *T* increases and can easily cause underflow if care is not taken. To solve this problem, we can scale the forward and backward variables while maintaining the validity of the re-estimation formulae.

The scaled version of the Forward algorithm normalizes each $\alpha_t(i)$ by dividing by the sum (over *j*) of all $\alpha_t(j)$ for each value *t*, or observation O_t. Let $\tilde{\alpha}_t(i)$ denotes the forward probability that is scaled up to $t-1$ but not scaled for *t* yet; $\hat{\alpha}_t(i)$ denotes the scaled probability; and $\alpha_t(i)$ denotes the non-scaled probability as given in the original forward algorithm. The scaling coefficient c_t at each time *t* is defined by

$$c_t = \frac{1}{\sum_{j=0}^{N-1} \tilde{\alpha}_t(j)},$$

where $c_0 = \frac{1}{\sum_{j=0}^{N-1} \alpha_0(j)}$ and $\hat{\alpha}_0(i) = c_0 \alpha_0(i)$ for $i = 0, 1, \ldots, N-1$ when $t = 0$.

Then for each $t = 1, 2, \ldots, T-1$, calculate

$$\tilde{\alpha}_t(i) = \sum_{j=0}^{N-1} \hat{\alpha}_{t-1}(j) a_{ji} b_i(O_t) \text{ and}$$

$$\hat{\alpha}_t(i) = c_t \tilde{\alpha}_t(i) \qquad \text{for } i = 0, 1, \ldots, N-1.$$

The scaled probabilities are now normalized so that $\sum_{i=0}^{N-1} \hat{\alpha}_t(i) = 1$. Also, it can be proven by induction that

$$\hat{\alpha}_t(i) = c_t \tilde{\alpha}_t(i)$$

$$= c_0 c_1 \ldots c_t \alpha_t(i).$$

Combining these two properties and setting $t = T-1$, we have

$$1 = \sum_{j=0}^{N-1} \hat{\alpha}_{T-1}(j)$$

$$\Leftrightarrow \quad 1 = c_0 c_1 \ldots c_{T-1} \sum_{j=0}^{N-1} \alpha_{T-1}(j)$$

$$\Leftrightarrow \quad 1 = c_0 c_1 ... c_{T-1} P(O \mid \lambda)$$

$$\Leftrightarrow \quad P(O \mid \lambda) = \frac{1}{\prod_{j=0}^{T-1} c_j}.$$

To avoid underflow, we compute the log likelihood $\log[P(O \mid \lambda)]$, instead of the $P(O \mid \lambda)$:

$$\log[P(O \mid \lambda)] = \log \frac{1}{\prod_{j=0}^{T-1} c_j}$$

$$= -\sum_{j=0}^{T-1} \log c_j .$$

The same scale factor c_t is used for $\beta_t(i)$ so that $\hat{\beta}_t(i) = c_t \beta_t(i)$. The computations of $\gamma_t(i)$ and $\gamma_t(i, j)$ use the same formulae as given in Section 4.1.2.3 substituting $\hat{\alpha}_t(i)$ and $\hat{\beta}_t(i)$ for $\alpha_t(i)$ and $\beta_t(i)$. These values are then used to re-estimate the model parameters A, B, and π.

The implementation of the Viterbi algorithm can also result in underflow. This is avoided by taking logarithms. The underflow-resistant Viterbi algorithm is defined as:

Step 1 Initialization:

$$\hat{\delta}_0(i) = \log[\pi_i b_i(O_0)], \qquad \text{for } i = 0, 1, \ldots, N-1$$

Step 2 Induction:

$$\hat{\delta}_t(i) = \max_{0 \le j \le N-1} \{\hat{\delta}_{t-1}(j) + \log[a_{ji}] + \log[b_i(O_t)]\},$$

for $t = 1, 2, \ldots, T-1$ and $i = 0, 1, \ldots, N-1$.

The optimal log probability is given by

$$\log P^* = \max_{0 \le i \le N-1} [\hat{\delta}_{T-1}(i)]$$

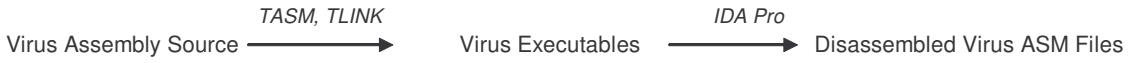
and as before back-pointers can be used to keep track of the optimal path.

4.2 HMM for Computer Virus Detection

Given a set of metamorphic virus variants, our goal is to train one or more hidden Markov models (HMMs) to represent the statistical properties of the virus family so that we can later use a trained model to determine whether a given program is similar to the viruses in the training set.

We trained our models based on the assembly opcode sequences of the virus files. For viruses originally generated in assembly source format, we first compiled the assembly source into executables using TASM 5.0. We then disassembled the executables using IDA Pro with identical default settings. We trained our models on the IDA-generated files rather than the original assembly source from the virus generators. We believed this makes our method more realistic. Disassembling executables is typically part of the virus

analysis process. This virus pre-processing procedure is the same as the one we used in the virus similarity test in Section 3 and is summarized again below:



There are generally two approaches to training an HMM when there are multiple observation sequences. We can either concatenate the sequences and make them into one long observation sequence; or train the HMM with each sequence separately and average the parameters from the different trainings [4]. We chose the former approach in our training process. With the set of pre-processed virus ASM files, we extracted the assembly opcode sequences, concatenated them into one long sequence of opcodes and used it to train our HMMs.

A trained model maximizes the probability of observing the training sequence. By calculating the probability of observing any given sequence in the HMM and comparing it to the probability of observing the training sequence, we know how well the given sequence matches the training sequence, or how "similar" the given sequence is to the training sequence. When trained with multiple sequences, the resulting HMM represents the "average" behavior, or the behavior of all the sequences in the form of a statistical profile. We can represent a whole virus family, as opposed to individual viruses, with a single HMM. The probability of any sequence in the HMM then tells us how likely it is that the given sequence belongs to the same virus family.

One extremely useful aspect of an HMM is that it tells us something about the training sequence without any requirement that we know how to interpret the observations or underlying features. Without specific knowledge of the features of the metamorphic viruses, we trained our HMMs using different number of states and examined the resulting probabilities to deduce what features the states represent. The number of states N that we tested are $N = 2, 3, 4, 5$, and 6. The number of observation symbols (opcodes) M, varies from model to model. We set M equal to the total number of distinct opcodes seen in all the training sequences for each model. With our data, M was typically around 70 to 80. The viruses we trained on have about 350 to 450 opcodes each, with an average of 416 opcodes. Concatenating 160 virus opcodes to train a model made the length of the observed training sequence T in the range of 66,000 to 67,000. The average T for the models we trained is 66,650.

Our HMM implementation used the scaled version of the Forward and the Backward algorithm as discussed in Section 4.1.3. To avoid underflow, we computed the log likelihood, instead of the raw probability, of observing the training sequence in the model at each step of the iterative training process. Re-estimation stopped when the log likelihood of the training sequence converged or a maximum of 800 iterations have been reached.

4.3 Training and Testing

Training and testing was done using standard cross-validation methodology [6]. With five-fold cross validation, we divide the data set into five equal-sized subsets. Each time when we train a model, we choose one of the subsets as the test set and train the model using data from the other four subsets. Because data from the test set is not used during training, we can use it to evaluate the performance of the model over unseen instances. Repeating this process five times, choosing a different subset as the test set each time, we can get five different models from the same set of data.

After training, a model should assign high probabilities to files similar to the training viruses and low probabilities to all other files, whether they are viruses from different families or "normal" benign programs. We made a comparison set which consisted of viruses generated by creation kits other than the one used to generate the training viruses and normal executable files of sizes comparable to the virus executables. We computed the log likelihood of the virus variants in the test set and the other programs in the comparison set using a trained model. Log likelihood is strongly length dependent, since it is a sum of log transition probabilities and log observation probabilities. A longer sequence will naturally have more transitions and more observations and thus a greater log likelihood, independent of how similar it is to the training sequences. Because the sequences in the comparison set may have lengths different from the sequences in the training and test set, we divided the log likelihood of a sequence by the sequence length

(which is the number of opcodes) to obtain the *log likelihood per opcode* (*LLPO*), which adjusts for the length difference. This LLPO is the score of the sequence.

With a trained model, we scored the files in the test set and those in the comparison set. There should be a separation of scores between files from these two sets as the model should assign higher probabilities and thus higher log likelihood per opcode to files in the same virus family. From these empirical scores, we determined a threshold, above which we will consider a file as belonging to the same family as the viruses in the training set. To classify whether a program is in the same virus family as the training data, we compute its score and compare it to the threshold.

The training and classifying process is summarized as

Training:

1) Given a data set consisting of different variants of a metamorphic virus, pick one subset as the test set and use the remaining four subsets for training.
2) Train HMM λ for sequences in the training set until the log likelihood of the training sequence converges.
3) Compute the score, i.e., the log probability per opcode (LLPO), of viruses in the test set and other files in the comparison set.
4) Determine a cutpoint (threshold) score above which a file is classified as a member virus. The threshold separates virus family members from non-members.

5) Repeat from (1), choosing a different subset as the test set, until all five subsets have been chosen.

Classifying:

1) To determine whether any program is part of the virus family, score and compare its LLPO to the model thresholds.

The HMM algorithms were implemented in C and compiled with Visual C++ 2005 Express Edition. We wrote some Ruby scripts using Ruby 1.8.4 on Windows [13] to perform the cross-validation. All trainings are carried out on a Pentium M 1.4 GHz machine running Windows XP Home Edition with 768 MB of RAM.

4.4 Data Used

Our data set consisted of 200 viruses generated by the Next Generation Virus Creation Kit (NGVCK), which was shown to be the most effective of the four virus generators we tested in Section 3. With five-fold cross validation, the number of viruses in each test set was 40 and the number of sequences used for training was 160 for each of the model. There were 65 files in the comparison set consisting of both viral and benign programs. These included:

- 25 viruses generated by the three generators G2, MPCGEN, and VCL32. They were the same programs that we tested for similarity in Section 3;
- 40 random executable files chosen from the Cygwin DLL (version 1.5.19) to represent "normal" benign programs. The first 20 were the same ones that we used in our similarity test.

All these programs were unique and there were no duplicates. Training and testing used files disassembled by IDA Pro (version 4.6.0) [3]. The four generators are downloadable from [19] while the Cygwin DLL is available at [2].

The IDA-preprocessed files were named as follows:

- the 200 viruses in the data set were named IDA_N0 to IDA_N199 (N for NGVCK);
- the 25 "other" viruses in the comparison set were named IDA_V0 to IDA_V24 (V for viruses);
- the 40 "normal" files were named IDA_R0 to IDA_R39 (R for random).

We divided the 200 viruses in the data set into five subsets of 40 viruses according to virus number:

- Test set 0: IDA_N0 to IDA_N39;
- Test set 1: IDA_N40 to IDA_N79;
- Test set 2: IDA_N80 to IDA_N119;
- Test set 3: IDA_N120 to IDA_N159;

- Test set 4: IDA_N160 to IDA_N199.

4.5 Experiment Result

For each N = 2, 3, 4, 5, and 6, training and testing was run as described above and five models were obtained for each N giving a total of 25 models. Seven of the models made a complete separation of scores between viruses in the test set and files in the comparison set. That is, the log likelihood per opcode (LLPO) of the family viruses were all higher than those of the non-family viruses and the random files. For the other models, we find some overlapping of scores where some non-family viruses have scores higher than some of the family viruses.

Figure 10 shows the result of a test run. The model in this test had three states, i.e., N = 3, and used test set 0 as the test set. As can be seen in Figure 10, for this case all random files have scores lower than those of the family viruses in the test set. However, the score distinction between family viruses and non-family viruses is not as clear. Some non-family viruses in the comparison set have scores very close to or higher than the family viruses. In the example shown here, the separation between family and non-family viruses is not perfect.

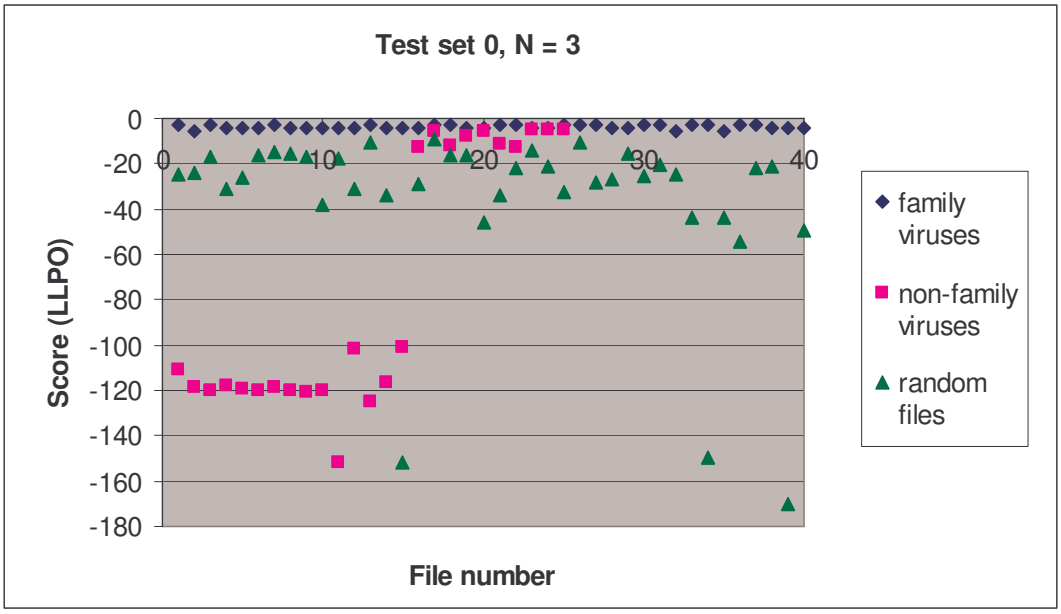


Figure 10 Log likelihood per opcode (LLPO) of family viruses, non-family viruses and random files. This test used test set 0 and N = 3.

The result illustrated in Figure 10 is typical of most models. In fact, if we look at the graph for each of the test sets for each *N*, the score distribution is very similar. If a file has a low score in one model, it always has a low score in all other models, although the scores are not always identical. We have included more of these graphs in Appendix B. Table B-1 shows the models trained with $N = 3$ states and Table B-2 shows the models with $N = 5$ states. The shapes of the curves are very similar in every graph. Our HMMs show consistent performance over the test data, regardless of number of hidden states. The raw scores of all the test runs are listed in Table B-3 in Appendix B.

Next we want to quantify the numbmer of *false positives* and *false negatives* associated with each model. A false positive occurs when a program not belonging to the virus family represented by an HMM is classified by the HMM as being a member virus. A false negative occurs when a member virus is misclassified as being a non-member. Analogously, true positives are family viruses correctly classified as members; while true negatives are programs not belonging to the virus family correctly classified as non-members.

Recall that a trained HMM classifies a program by comparing its log likelihood per opcode (LLPO) to the threshold LLPO. The choice of threshold value therefore affects the classification and thus the amount of false positives and false negatives a model produces. If we choose a higher threshold, fewer programs would score higher than the threshold and there would be fewer false positives. This, however, is usually accompanied by more false negatives as more member viruses may have scores lower than the threshold. We examined the amount of false positives and false negatives that came with different threshold values. Figure 11 illustrates the tradeoff between the two when the threshold changes from -3.5 to -2.5, for the model with $N = 2$ hidden states using test set 4.

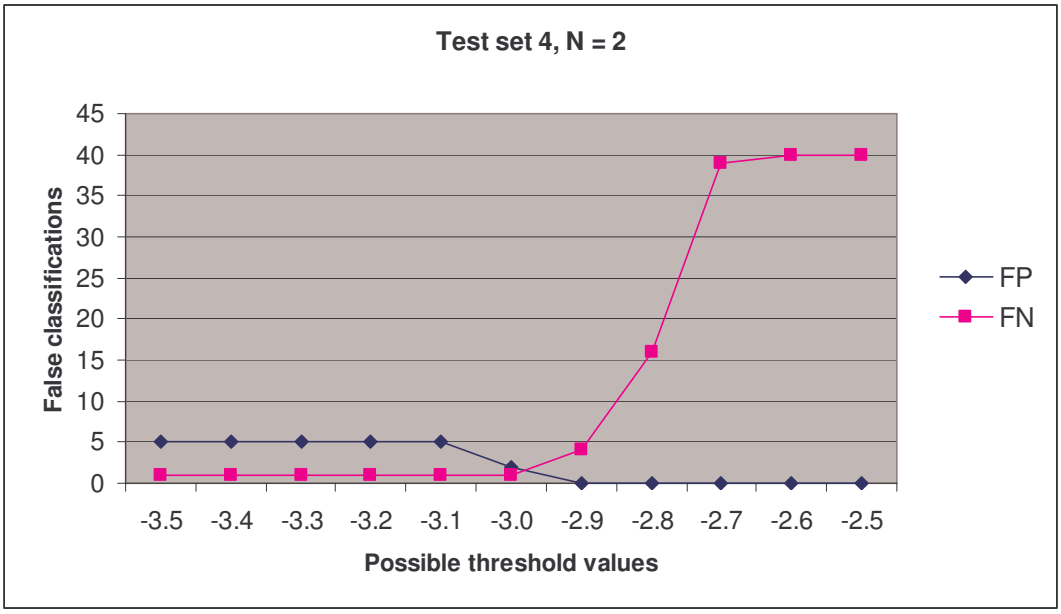


Figure 11 The tradeoff between false positives (FP) and false negatives (FN) with changing threshold values.

Table 5 shows the number of false classifications as plotted in Figure 11. At a threshold value of -3.0, there are two false positives and one false negative. Increasing the threshold to -2.9 reduces the number of false positives to zero but increases the number of false negatives to four. Depending on the desired tradeoff, we could select the threshold accordingly.

Threshold	-3.5	-3.4	-3.3	-3.2	-3.1	-3.0	-2.9	-2.8	-2.7	-2.6	-2.5
FP	5	5	5	5	5	2	0	0	0	0	0
FN	1	1	1	1	1	1	4	16	39	40	40

Table 5 False positive (FP) and false negative (FN) counts for threshold ranging from -3.5 to -2.5. This model used test set 4 and N = 2.

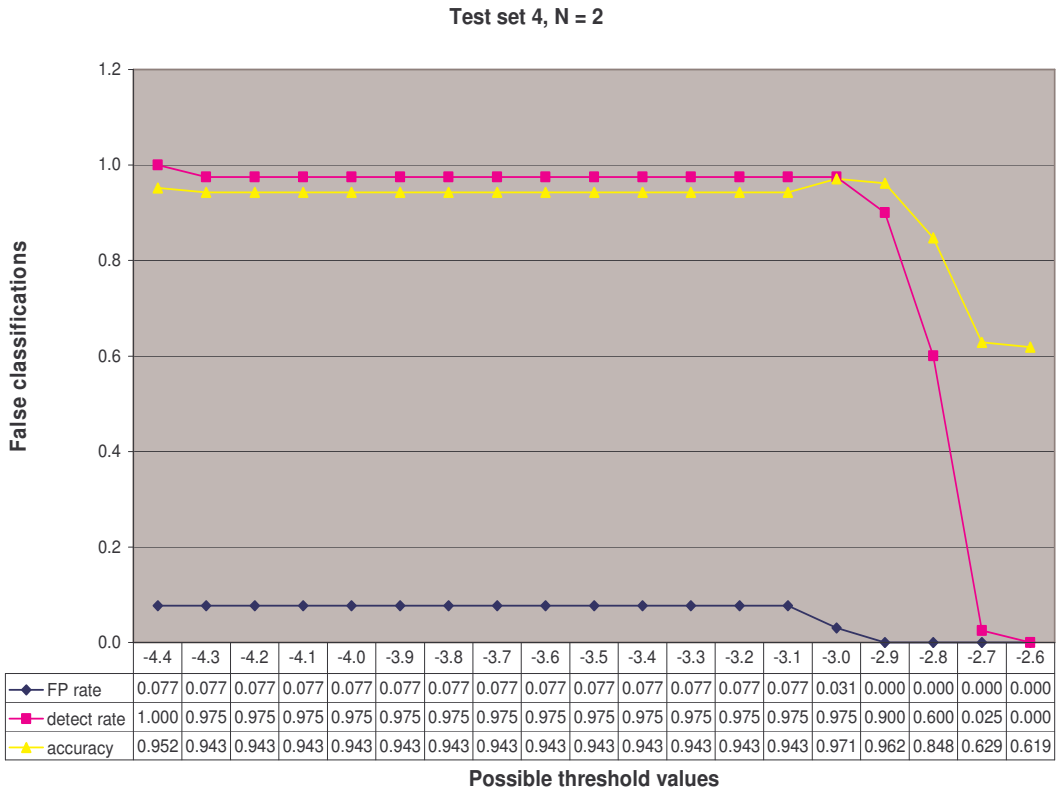
Besides the raw false positive and false negative counts, we calculated three other performance measures based on these counts: *detection rate*, *false positive rate*, and *overall accuracy*. The detection rate tells us the sensitivity of the model and is defined as the number of member viruses that are caught by an HMM divided by the total number of member viruses in the test set (40 in our experiments). The false positive rate is related to the specificity of the model and is defined as the number of false positives divided by the total number of non-member programs in the comparison set (65 in our test runs). Overall accuracy is defined as the number of true predictions (positives and negatives) divided by the total number of member and non-member programs (105 in our tests). The three measures are related to true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) as follows:

- $\textit{Detection rate} = \frac{TP}{TP + FN}$, as TP + FN equals total number of member viruses tested;
- $\textit{False positive rate} = \frac{FP}{FP + TN}$, as FP + TN equals total number of non-member programs tested;
- $\textit{Overall accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$.

The detection rate, false positive rate, and overall accuracy of the test run above are shown in Figure 12. We plotted the rates from threshold -4.5 to -2.5. The three rates are again functions of the threshold. At a threshold value of -3.0, the detection rate and

overall accuracy are 97.5% and 97.1% respectively while the false positive rate is 3.1%. If we increase the threshold to -2.9, the false positive rate would be 0% but both detection rate and accuracy would drop, to 90% and 96.2% respectively.

There is no one best way to define what an “optimal” threshold is. If false alarms are absolutely unacceptable, we may want to choose the higher threshold of -2.9. On the other hard, if it is absolutely necessary that we detect all viruses in the family, we may need to use the lower threshold and we may go all the way to the threshold of -4.4 which gives a 100% detection rate. This of course comes at the expense of a higher false positive rate, which is 7.7% in this case. Choosing a threshold always implies a tradeoff.



	-4.4	-4.3	-4.2	-4.1	-4.0	-3.9	-3.8	-3.7	-3.6	-3.5	-3.4	-3.3	-3.2	-3.1	-3.0	-2.9	-2.8	-2.7	-2.6
FP rate	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.077	0.031	0.000	0.000	0.000	0.000
detect rate	1.000	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.975	0.900	0.600	0.025	0.000
accuracy	0.952	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.943	0.971	0.962	0.848	0.629	0.619

Figure 12 Comparison of false positive rate, detection rate and overall accuracy.

Suppose we want to limit the false negative rate to 10%. In other words, we want to have a detection rate of 90% or more. The threshold values that would produce the desired detection performance are listed in Table 6. The value for each model is the largest threshold LLPO that can still maintain a false negative rate of 10%. If we choose a threshold lower than the listed value, it is possible to achieve a higher detection rate, although it is likely that the increase in detection rate will be accompanied by an increase in false positive rate. The false positive rates for all the models, at the respective

threshold values, fall within 0% to 7.7%. All models with three states (N = 3) produced 0% false positives. Even so, the overall accuracy of all the models is quite similar.

Detection rate >= 90%		threshold	FP	FN	detect rate	FP rate	accuracy
N = 2	test 0	**-3.0**	2	3	0.925	0.031	0.952
	test 1	**-2.9**	2	4	0.900	0.031	0.943
	test 2	**-2.9**	1	3	0.925	0.015	0.962
	test 3	**-4.4**	5	2	0.950	0.077	0.933
	test 4	**-2.9**	0	4	0.900	0.000	0.962
N = 3	test 0	**-4.5**	0	4	0.900	0.000	0.962
	test 1	**-4.4**	0	3	0.925	0.000	0.971
	test 2	**-2.8**	0	4	0.900	0.000	0.962
	test 3	**-4.3**	0	4	0.900	0.000	0.962
	test 4	**-2.8**	0	4	0.900	0.000	0.962
N = 4	test 0	**-2.8**	0	3	0.925	0.000	0.971
	test 1	**-2.7**	0	4	0.900	0.000	0.962
	test 2	**-2.7**	2	4	0.900	0.031	0.943
	test 3	**-4.2**	3	4	0.900	0.046	0.933
	test 4	**-2.7**	0	4	0.900	0.000	0.962
N = 5	test 0	**-2.7**	0	4	0.900	0.000	0.962
	test 1	**-2.7**	3	4	0.900	0.046	0.933
	test 2	**-2.7**	0	4	0.900	0.000	0.962
	test 3	**-4.2**	5	3	0.925	0.077	0.924
	test 4	**-2.7**	0	3	0.925	0.000	0.971
N = 6	test 0	**-2.7**	0	4	0.900	0.000	0.962
	test 1	**-4.2**	0	3	0.925	0.000	0.971
	test 2	**-4.1**	5	4	0.900	0.077	0.914
	test 3	**-4.2**	3	1	0.975	0.046	0.962
	test 4	**-2.6**	0	4	0.900	0.000	0.962

Table 6 Threshold LLPO that gives a detection rate of 90% or more for each model.

Next, we pick the value -4.5, which is the lowest threshold in the analysis above, and see how the performance measures would change with this lower threshold value. Table 7 shows the false positive count, false negative count, detection rate, false positive rate and overall accuracy when we set the cutpoint at -4.5 for all the models. Compared to the

previous table, the detection rates as well as the false positive rates indeed have increased for most models. We see that 17 of the models have detection rate reaching 100% and 10 models have 0% false positive rate. Although the performance of all the models is quite similar, models with two states (N = 2) do have slightly higher false positive rates and lower accuracy. We conclude there is not a significant difference in performance between models with three or more states. With the right choice of threshold, detection rate and overall accuracy of all models can reach 90% or more while the false positive rate can be kept below 10%.

Threshold = -4.5		FP	FN	detect rate	FP rate	accuracy
N = 2	test 0	5	0	1.000	0.077	**0.952**
	test 1	5	2	0.950	0.077	**0.933**
	test 2	5	2	0.950	0.077	**0.933**
	test 3	5	0	1.000	0.077	**0.952**
	test 4	5	0	1.000	0.077	**0.952**
N = 3	test 0	0	4	0.900	0.000	**0.962**
	test 1	0	2	0.950	0.000	**0.981**
	test 2	0	1	0.975	0.000	**0.990**
	test 3	0	0	1.000	0.000	**1.000**
	test 4	0	0	1.000	0.000	**1.000**
N = 4	test 0	0	0	1.000	0.000	**1.000**
	test 1	3	2	0.950	0.046	**0.952**
	test 2	5	0	1.000	0.077	**0.952**
	test 3	3	0	1.000	0.046	**0.971**
	test 4	3	0	1.000	0.046	**0.971**
N = 5	test 0	0	0	1.000	0.000	**1.000**
	test 1	5	2	0.950	0.077	**0.933**
	test 2	5	0	1.000	0.077	**0.952**
	test 3	5	0	1.000	0.077	**0.952**
	test 4	0	0	1.000	0.000	**1.000**
N = 6	test 0	0	0	1.000	0.000	**1.000**
	test 1	0	3	0.925	0.000	**0.971**
	test 2	5	0	1.000	0.077	**0.952**
	test 3	3	0	1.000	0.046	**0.971**
	test 4	5	0	1.000	0.077	**0.952**

Table 7 False positive count, false negative count, detection rate, false positive rate and overall accuracy when threshold is set at -4.5 for all models.

5. CONCLUSION

Virus writers and anti-virus researches generally agree that metamorphism is the way to generate undetectable viruses. Several virus writers have released virus creation kits and claimed that they possess the ability to automatically produce morphed virus variants that look substantially different from one another.

To see how effective these code morphing engines are, and how much difference exists between variants of a same virus, we measured the similarity between virus variants generated by four virus generators downloaded from the Internet. Our result shows that the effectiveness of these generators varies widely. While the best generator, NGVCK, is able to create viruses that share only a few percent of similarities, the other generators produce viruses that are over 60% similar, on average. In addition, our similarity graphs show that some of these variant pairs have long segments of identical assembly opcodes at identical positions of the virus files. Compared to random utility files which have a similarity of about 35%, we see that some of the virus creation kits are not very effective. But since they produce assembly code, hackers with some knowledge of assembly programming can manually change the code further and make the virus more difficult to detect.

To detect metamorphic virus variants, we experimented with the use of hidden Markov models (HMMs) to capture the statistical properties of viruses in the same family. We generated 200 NGVCK viruses, trained 25 models and used the trained models to classify 65 programs including both NGVCK viruses and other random non-viral programs. For seven of our models we were able to perfectly distinguish the two types of files by their scores. The other cases produced different number of false positives and false negatives, depending on the threshold used in the classifying process. In most cases, our models were able to have a detection rate of over 90% and a false positive rate of less than 10%. The number of states N of a model does not seem to have much impact on the

performance of the HMM. We saw only small differences in the performance measures for models with *N* from 3 to 6.

If the variants of a metamorphic virus are sufficiently different that signature-based scanning cannot detect a newly morphed variant, the HMM approach provides a feasible solution. As with any statistical detection method, false predictions are possible. We showed the tradeoff between the detection rate and false positive rate. Of course, users generally want a classifier that has low false positive rate and low false negative rate. They would not want false alarms, which can be annoying and can destroy their trust to the classifier; nor would they want a model that may let viruses go undetected. Compromising between false positives and false negatives is a challenge with any statistical approach.

6. FUTURE WORK

We trained our models on disassembled virus executables. The disassembling process can take some time and the result of the disassembling depends on the quality of the disassembler. To speed up virus pre-processing and to eliminate the reliance on a particular disassembler, we can train the HMMs on the binary code of the viruses. Other machine learning techniques, like the data mining or the neural network approach, might also work directly on the binaries.

Training on raw executable byte sequences is more challenging as these byte sequences are longer and contain more irrelevant parts. We can train our HMMs on the complete code if the size of the virus is small enough. Otherwise, we can train the models using only the code segments and perhaps the data segments, excluding header and other kinds of identification information, since the behavior of a program is primarily determined by its code segments.

Our models were tested on only one family of metamorphic viruses, namely the viruses generated by the Next Generation Virus Creation Virus Creation Kit (NGVCK). To more thoroughly evaluate the performance of the HMM approach, it would be useful to test on a larger set of virus variants and also test on different types of viruses.

Bibliography

[1] W. Arnold, G. Tesauro, “Automatically Generated Win32 Heuristic Virus Detection”, *Proceedings of the 2000 International Virus Bulletin Conference*, 2000.

[2] Cygwin <http://cygwin.com/>

[3] IDA Pro Disassembler <http://www.datarescue.com/idabase/>

[4] R.S. Jensen, “Immune System for Virus Detection and Elimination”, master’s thesis, Informatics and Mathematical Modelling, Technical University of Denmark, IMM-EP-2002-55, 2002.
< www2.imm.dtu.dk/pubdb/views/edoc_download.php/959/pdf/imm959.pdf >

[5] J. Kephart, A. William, “Automatic Extraction of Computer Virus Signatures”, Proceedings of the 4th International Virus Bulletin Conference, R. Ford, ed., Virus Bulletin Ltd., Abingdon, England, pp. 178-184, 1994.
<http://www.research.ibm.com/antivirus/SciPapers/Kephart/VB94/vb94.html>

[6] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection”, *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pp. 1137-1143, 1995.

[7] A. Krogh, “An introduction to hidden Markov models for biological sequences”, *Computational Methods in Molecular Biology*, pp. 45-63, Elsevier, 1998.

[8] A. Krogh, M. Brown, I.S. Mian, K. Sjolander, D. Haussler, “Hidden markov models in computational biology: applications to protein modeling”, *J. Mol. Biol.*, vol. 235, no. 5, pp. 1501-1531, 1994.

[9] P. Mishra, “A taxonomy of software uniqueness transformations”, master’s thesis, San Jose State University, Dec. 2003.
<http://home.earthlink.net/~mstamp1/mss_v.html#masters>

[10] M. Mohammed, “Zeroing in on metamorphic computer viruses”, master’s thesis, University of Louisiana at Lafayette, Dec. 2003.
<http://www.cacs.louisiana.edu/~arun/papers/moin-mohammed-thesis-dec2003.pdf>

[11] I. Muttik, “Silicon Implants”, *Virus Bulletin*, pp. 8-10, May 1997.

[12] L.R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition”, *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[13] Ruby <http://www.ruby-lang.org/en/20020102.html>

[14] M.G. Schultz, E. Eskin, E. Zadok, S.J. Stolfo, "Data Mining Methods for Detection of New Malicious Executables", *sp*, pp. 0038, IEEE Symposium on Security and Privacy, 2001.

[15] M. Stamp, “A Revealing Introduction to Hidden Markov Models”, January 2004.
<http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf>

[16] P. Szor, *The Art of Computer Virus Research and Defense*, Addison-Wesley, 2005.

[17] P. Szor, P. Ferrie, “Hunting for Metamorphic”, *Symantec Security Response*. <http://enterprisesecurity.symantec.com/PDF/metamorphic.pdf>

[18] G. Tesauro, J.O. Kephart, G.B. Sorkin, “Neural networks for computer virus recognition”, *IEEE Expert*, vol. 11, no. 4, pp. 5-6, Aug. 1996. <http://www.research.ibm.com/antivirus/SciPapers/Tesauro/NeuralNets.html>

[19] VX Heavens. <http://vx.netlux.org/>

[20] washingtonpost.com Staff Writer, “A Short History of Computer Viruses and Attacks”, Feb. 2003. <http://www.washingtonpost.com/wp-dyn/articles/A50636-2002Jun26.html>

[21] Zombie, “About Permutation”, documentation of RPME permutation engine. <http://vx.netlux.org/vx.php?id=er05>

Appendix A: Virus similarity test results

Table A-1 Similarity scores between NGVCK virus variants.

Similarity scores between files:										
IDA_NGVCK0	IDA_NGVCK1	0.07434	IDA_NGVCK3	IDA_NGVCK13	0.10067	IDA_NGVCK8	IDA_NGVCK11	0.07875	min	0.02934
IDA_NGVCK0	IDA_NGVCK2	0.08920	IDA_NGVCK3	IDA_NGVCK14	0.10554	IDA_NGVCK8	IDA_NGVCK12	0.03634	max	0.17176
IDA_NGVCK0	IDA_NGVCK3	0.15131	IDA_NGVCK3	IDA_NGVCK15	0.08981	IDA_NGVCK8	IDA_NGVCK13	0.03600	average	0.09600
IDA_NGVCK0	IDA_NGVCK4	0.18340	IDA_NGVCK3	IDA_NGVCK16	0.13886	IDA_NGVCK8	IDA_NGVCK14	0.02934		
IDA_NGVCK0	IDA_NGVCK5	0.09070	IDA_NGVCK3	IDA_NGVCK17	0.14873	IDA_NGVCK8	IDA_NGVCK15	0.07818		
IDA_NGVCK0	IDA_NGVCK6	0.05134	IDA_NGVCK3	IDA_NGVCK18	0.13848	IDA_NGVCK8	IDA_NGVCK16	0.04610		
IDA_NGVCK0	IDA_NGVCK7	0.05413	IDA_NGVCK3	IDA_NGVCK19	0.12308	IDA_NGVCK8	IDA_NGVCK17	0.04854		
IDA_NGVCK0	IDA_NGVCK8	0.11911	IDA_NGVCK4	IDA_NGVCK5	0.08773	IDA_NGVCK8	IDA_NGVCK18	0.06508		
IDA_NGVCK0	IDA_NGVCK9	0.09770	IDA_NGVCK4	IDA_NGVCK6	0.10706	IDA_NGVCK8	IDA_NGVCK19	0.13540		
IDA_NGVCK0	IDA_NGVCK10	0.12208	IDA_NGVCK4	IDA_NGVCK7	0.11275	IDA_NGVCK9	IDA_NGVCK10	0.15118		
IDA_NGVCK0	IDA_NGVCK11	0.17967	IDA_NGVCK4	IDA_NGVCK8	0.07676	IDA_NGVCK9	IDA_NGVCK11	0.11877		
IDA_NGVCK0	IDA_NGVCK12	0.14436	IDA_NGVCK4	IDA_NGVCK9	0.09182	IDA_NGVCK9	IDA_NGVCK12	0.09489		
IDA_NGVCK0	IDA_NGVCK13	0.10156	IDA_NGVCK4	IDA_NGVCK10	0.18537	IDA_NGVCK9	IDA_NGVCK13	0.13758		
IDA_NGVCK0	IDA_NGVCK14	0.12691	IDA_NGVCK4	IDA_NGVCK11	0.05152	IDA_NGVCK9	IDA_NGVCK14	0.09824		
IDA_NGVCK0	IDA_NGVCK15	0.09563	IDA_NGVCK4	IDA_NGVCK12	0.10682	IDA_NGVCK9	IDA_NGVCK15	0.11261		
IDA_NGVCK0	IDA_NGVCK16	0.13088	IDA_NGVCK4	IDA_NGVCK13	0.06559	IDA_NGVCK9	IDA_NGVCK16	0.16471		
IDA_NGVCK0	IDA_NGVCK17	0.09841	IDA_NGVCK4	IDA_NGVCK14	0.17728	IDA_NGVCK9	IDA_NGVCK17	0.07887		
IDA_NGVCK0	IDA_NGVCK18	0.12794	IDA_NGVCK4	IDA_NGVCK15	0.13155	IDA_NGVCK9	IDA_NGVCK18	0.10710		
IDA_NGVCK0	IDA_NGVCK19	0.07873	IDA_NGVCK4	IDA_NGVCK16	0.10552	IDA_NGVCK9	IDA_NGVCK19	0.15248		
IDA_NGVCK1	IDA_NGVCK2	0.08636	IDA_NGVCK4	IDA_NGVCK17	0.10273	IDA_NGVCK10	IDA_NGVCK11	0.10869		
IDA_NGVCK1	IDA_NGVCK3	0.10922	IDA_NGVCK4	IDA_NGVCK18	0.07407	IDA_NGVCK10	IDA_NGVCK12	0.17176		
IDA_NGVCK1	IDA_NGVCK4	0.16578	IDA_NGVCK4	IDA_NGVCK19	0.11025	IDA_NGVCK10	IDA_NGVCK13	0.08110		
IDA_NGVCK1	IDA_NGVCK5	0.09711	IDA_NGVCK5	IDA_NGVCK6	0.05343	IDA_NGVCK10	IDA_NGVCK14	0.15890		
IDA_NGVCK1	IDA_NGVCK6	0.12297	IDA_NGVCK5	IDA_NGVCK7	0.07103	IDA_NGVCK10	IDA_NGVCK15	0.16645		
IDA_NGVCK1	IDA_NGVCK7	0.09787	IDA_NGVCK5	IDA_NGVCK8	0.12342	IDA_NGVCK10	IDA_NGVCK16	0.12996		
IDA_NGVCK1	IDA_NGVCK8	0.07977	IDA_NGVCK5	IDA_NGVCK9	0.12222	IDA_NGVCK10	IDA_NGVCK17	0.11580		
IDA_NGVCK1	IDA_NGVCK9	0.19684	IDA_NGVCK5	IDA_NGVCK10	0.07149	IDA_NGVCK10	IDA_NGVCK18	0.06672		
IDA_NGVCK1	IDA_NGVCK10	0.17116	IDA_NGVCK5	IDA_NGVCK11	0.12851	IDA_NGVCK10	IDA_NGVCK19	0.04028		
IDA_NGVCK1	IDA_NGVCK11	0.10572	IDA_NGVCK5	IDA_NGVCK12	0.06257	IDA_NGVCK11	IDA_NGVCK12	0.05686		
IDA_NGVCK1	IDA_NGVCK12	0.11574	IDA_NGVCK5	IDA_NGVCK13	0.03453	IDA_NGVCK11	IDA_NGVCK13	0.14430		
IDA_NGVCK1	IDA_NGVCK13	0.11579	IDA_NGVCK5	IDA_NGVCK14	0.05849	IDA_NGVCK11	IDA_NGVCK14	0.12858		
IDA_NGVCK1	IDA_NGVCK14	0.14021	IDA_NGVCK5	IDA_NGVCK15	0.05950	IDA_NGVCK11	IDA_NGVCK15	0.14992		
IDA_NGVCK1	IDA_NGVCK15	0.08796	IDA_NGVCK5	IDA_NGVCK16	0.05158	IDA_NGVCK11	IDA_NGVCK16	0.13306		
IDA_NGVCK1	IDA_NGVCK16	0.07606	IDA_NGVCK5	IDA_NGVCK17	0.10532	IDA_NGVCK11	IDA_NGVCK17	0.11945		
IDA_NGVCK1	IDA_NGVCK17	0.09617	IDA_NGVCK5	IDA_NGVCK18	0.06744	IDA_NGVCK11	IDA_NGVCK18	0.10001		
IDA_NGVCK1	IDA_NGVCK18	0.11478	IDA_NGVCK5	IDA_NGVCK19	0.16166	IDA_NGVCK11	IDA_NGVCK19	0.11414		
IDA_NGVCK1	IDA_NGVCK19	0.11744	IDA_NGVCK6	IDA_NGVCK7	0.07618	IDA_NGVCK12	IDA_NGVCK13	0.03950		
IDA_NGVCK2	IDA_NGVCK3	0.11767	IDA_NGVCK6	IDA_NGVCK8	0.06070	IDA_NGVCK12	IDA_NGVCK14	0.11242		
IDA_NGVCK2	IDA_NGVCK4	0.10050	IDA_NGVCK6	IDA_NGVCK9	0.10760	IDA_NGVCK12	IDA_NGVCK15	0.12866		
IDA_NGVCK2	IDA_NGVCK5	0.08412	IDA_NGVCK6	IDA_NGVCK10	0.15063	IDA_NGVCK12	IDA_NGVCK16	0.03688		
IDA_NGVCK2	IDA_NGVCK6	0.05393	IDA_NGVCK6	IDA_NGVCK11	0.07058	IDA_NGVCK12	IDA_NGVCK17	0.05149		
IDA_NGVCK2	IDA_NGVCK7	0.12356	IDA_NGVCK6	IDA_NGVCK12	0.08605	IDA_NGVCK12	IDA_NGVCK18	0.10002		
IDA_NGVCK2	IDA_NGVCK8	0.10744	IDA_NGVCK6	IDA_NGVCK13	0.06433	IDA_NGVCK12	IDA_NGVCK19	0.09563		
IDA_NGVCK2	IDA_NGVCK9	0.04529	IDA_NGVCK6	IDA_NGVCK14	0.08921	IDA_NGVCK13	IDA_NGVCK14	0.09217		
IDA_NGVCK2	IDA_NGVCK10	0.11901	IDA_NGVCK6	IDA_NGVCK15	0.03582	IDA_NGVCK13	IDA_NGVCK15	0.08607		
IDA_NGVCK2	IDA_NGVCK11	0.04575	IDA_NGVCK6	IDA_NGVCK16	0.07146	IDA_NGVCK13	IDA_NGVCK16	0.04954		
IDA_NGVCK2	IDA_NGVCK12	0.06784	IDA_NGVCK6	IDA_NGVCK17	0.15974	IDA_NGVCK13	IDA_NGVCK17	0.13265		
IDA_NGVCK2	IDA_NGVCK13	0.01493	IDA_NGVCK6	IDA_NGVCK18	0.08771	IDA_NGVCK13	IDA_NGVCK18	0.05564		
IDA_NGVCK2	IDA_NGVCK14	0.11570	IDA_NGVCK6	IDA_NGVCK19	0.05652	IDA_NGVCK13	IDA_NGVCK19	0.07022		
IDA_NGVCK2	IDA_NGVCK15	0.09738	IDA_NGVCK7	IDA_NGVCK8	0.10729	IDA_NGVCK14	IDA_NGVCK15	0.16591		
IDA_NGVCK2	IDA_NGVCK16	0.06714	IDA_NGVCK7	IDA_NGVCK9	0.09201	IDA_NGVCK14	IDA_NGVCK16	0.09793		
IDA_NGVCK2	IDA_NGVCK17	0.02224	IDA_NGVCK7	IDA_NGVCK10	0.17010	IDA_NGVCK14	IDA_NGVCK17	0.09638		
IDA_NGVCK2	IDA_NGVCK18	0.05040	IDA_NGVCK7	IDA_NGVCK11	0.12210	IDA_NGVCK14	IDA_NGVCK18	0.06559		
IDA_NGVCK2	IDA_NGVCK19	0.08155	IDA_NGVCK7	IDA_NGVCK12	0.04414	IDA_NGVCK14	IDA_NGVCK19	0.08164		
IDA_NGVCK3	IDA_NGVCK4	0.14915	IDA_NGVCK7	IDA_NGVCK13	0.08843	IDA_NGVCK15	IDA_NGVCK16	0.14119		
IDA_NGVCK3	IDA_NGVCK5	0.13363	IDA_NGVCK7	IDA_NGVCK14	0.21018	IDA_NGVCK15	IDA_NGVCK17	0.03772		
IDA_NGVCK3	IDA_NGVCK6	0.15358	IDA_NGVCK7	IDA_NGVCK15	0.17078	IDA_NGVCK15	IDA_NGVCK18	0.08714		
IDA_NGVCK3	IDA_NGVCK7	0.14616	IDA_NGVCK7	IDA_NGVCK16	0.09845	IDA_NGVCK15	IDA_NGVCK19	0.08801		
IDA_NGVCK3	IDA_NGVCK8	0.05070	IDA_NGVCK7	IDA_NGVCK17	0.11370	IDA_NGVCK16	IDA_NGVCK17	0.08680		
IDA_NGVCK3	IDA_NGVCK9	0.13307	IDA_NGVCK7	IDA_NGVCK18	0.08161	IDA_NGVCK16	IDA_NGVCK18	0.03431		
IDA_NGVCK3	IDA_NGVCK10	0.13738	IDA_NGVCK7	IDA_NGVCK19	0.14470	IDA_NGVCK16	IDA_NGVCK19	0.04922		
IDA_NGVCK3	IDA_NGVCK11	0.13700	IDA_NGVCK8	IDA_NGVCK9	0.12738	IDA_NGVCK17	IDA_NGVCK18	0.06581		
IDA_NGVCK3	IDA_NGVCK12	0.05351	IDA_NGVCK8	IDA_NGVCK10	0.10699	IDA_NGVCK17	IDA_NGVCK19	0.15762		
						IDA_NGVCK18	IDA_NGVCK19	0.08161		

Table A-2 Similarity scores between G2 virus variants.

Similarity scores between files:				
IDA_G0	IDA_G1	0.70808	min	0.62845
IDA_G0	IDA_G2	0.79452	max	0.84864
IDA_G0	IDA_G3	0.79818	average	0.74491
IDA_G0	IDA_G4	0.70615		
IDA_G0	IDA_G5	0.73516		
IDA_G0	IDA_G6	0.64831		
IDA_G0	IDA_G7	0.77626		
IDA_G0	IDA_G8	0.73685		
IDA_G0	IDA_G9	0.68037		
IDA_G1	IDA_G2	0.72647		
IDA_G1	IDA_G3	0.77599		
IDA_G1	IDA_G4	0.66519		
IDA_G1	IDA_G5	0.80004		
IDA_G1	IDA_G6	0.76389		
IDA_G1	IDA_G7	0.78624		
IDA_G1	IDA_G8	0.78343		
IDA_G1	IDA_G9	0.72187		
IDA_G2	IDA_G3	0.68350		
IDA_G2	IDA_G4	0.71527		
IDA_G2	IDA_G5	0.71690		
IDA_G2	IDA_G6	0.67589		
IDA_G2	IDA_G7	0.78995		
IDA_G2	IDA_G8	0.76888		
IDA_G2	IDA_G9	0.76256		
IDA_G3	IDA_G4	0.71857		
IDA_G3	IDA_G5	0.84864		
IDA_G3	IDA_G6	0.79908		
IDA_G3	IDA_G7	0.62845		
IDA_G3	IDA_G8	0.78621		
IDA_G3	IDA_G9	0.67891		
IDA_G4	IDA_G5	0.76994		
IDA_G4	IDA_G6	0.67437		
IDA_G4	IDA_G7	0.75171		
IDA_G4	IDA_G8	0.78997		
IDA_G4	IDA_G9	0.80183		
IDA_G5	IDA_G6	0.79544		
IDA_G5	IDA_G7	0.71690		
IDA_G5	IDA_G8	0.84669		
IDA_G5	IDA_G9	0.75799		
IDA_G6	IDA_G7	0.78165		
IDA_G6	IDA_G8	0.76960		
IDA_G6	IDA_G9	0.73567		
IDA_G7	IDA_G8	0.67735		
IDA_G7	IDA_G9	0.76256		
IDA_G8	IDA_G9	0.70939		

Table A-3 Similarity scores between VCL32 virus variants.

Similarity scores between files:				
IDA_VCL0	IDA_VCL1	0.66883	min	0.34376
IDA_VCL0	IDA_VCL2	0.71341	max	0.92907
IDA_VCL0	IDA_VCL3	0.40061	average	0.60631
IDA_VCL0	IDA_VCL4	0.81177		
IDA_VCL0	IDA_VCL5	0.63669		
IDA_VCL0	IDA_VCL6	0.80079		
IDA_VCL0	IDA_VCL7	0.41714		
IDA_VCL0	IDA_VCL8	0.56377		
IDA_VCL0	IDA_VCL9	0.60213		
IDA_VCL1	IDA_VCL2	0.43906		
IDA_VCL1	IDA_VCL3	0.65971		
IDA_VCL1	IDA_VCL4	0.81516		
IDA_VCL1	IDA_VCL5	0.38916		
IDA_VCL1	IDA_VCL6	0.57589		
IDA_VCL1	IDA_VCL7	0.69156		
IDA_VCL1	IDA_VCL8	0.85086		
IDA_VCL1	IDA_VCL9	0.79484		
IDA_VCL2	IDA_VCL3	0.79247		
IDA_VCL2	IDA_VCL4	0.55693		
IDA_VCL2	IDA_VCL5	0.91090		
IDA_VCL2	IDA_VCL6	0.64831		
IDA_VCL2	IDA_VCL7	0.34376		
IDA_VCL2	IDA_VCL8	0.35551		
IDA_VCL2	IDA_VCL9	0.38754		
IDA_VCL3	IDA_VCL4	0.50818		
IDA_VCL3	IDA_VCL5	0.72941		
IDA_VCL3	IDA_VCL6	0.44217		
IDA_VCL3	IDA_VCL7	0.52330		
IDA_VCL3	IDA_VCL8	0.53924		
IDA_VCL3	IDA_VCL9	0.49560		
IDA_VCL4	IDA_VCL5	0.47466		
IDA_VCL4	IDA_VCL6	0.55365		
IDA_VCL4	IDA_VCL7	0.51529		
IDA_VCL4	IDA_VCL8	0.70071		
IDA_VCL4	IDA_VCL9	0.74909		
IDA_VCL5	IDA_VCL6	0.58797		
IDA_VCL5	IDA_VCL7	0.49445		
IDA_VCL5	IDA_VCL8	0.51078		
IDA_VCL5	IDA_VCL9	0.56698		
IDA_VCL6	IDA_VCL7	0.62658		
IDA_VCL6	IDA_VCL8	0.46267		
IDA_VCL6	IDA_VCL9	0.41573		
IDA_VCL7	IDA_VCL8	0.85004		
IDA_VCL7	IDA_VCL9	0.78161		
IDA_VCL8	IDA_VCL9	0.92907		

Table A-4 Similarity scores between MPCGEN virus variants.

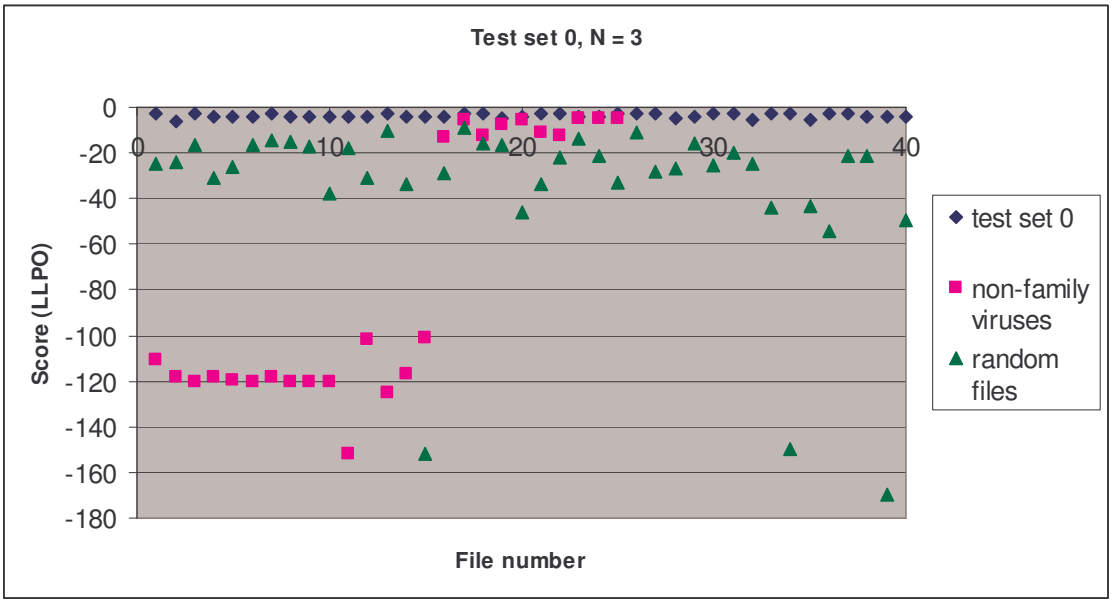
Similarity scores between files:				
IDA_MPC0	IDA_MPC1	0.45032	min	0.44964
IDA_MPC0	IDA_MPC2	0.46885	max	0.96568
IDA_MPC0	IDA_MPC3	0.78035	average	0.62704
IDA_MPC0	IDA_MPC4	0.44970		
IDA_MPC1	IDA_MPC2	0.80875		
IDA_MPC1	IDA_MPC3	0.57993		
IDA_MPC1	IDA_MPC4	0.96568		
IDA_MPC2	IDA_MPC3	0.44964		
IDA_MPC2	IDA_MPC4	0.80704		
IDA_MPC3	IDA_MPC4	0.51009		

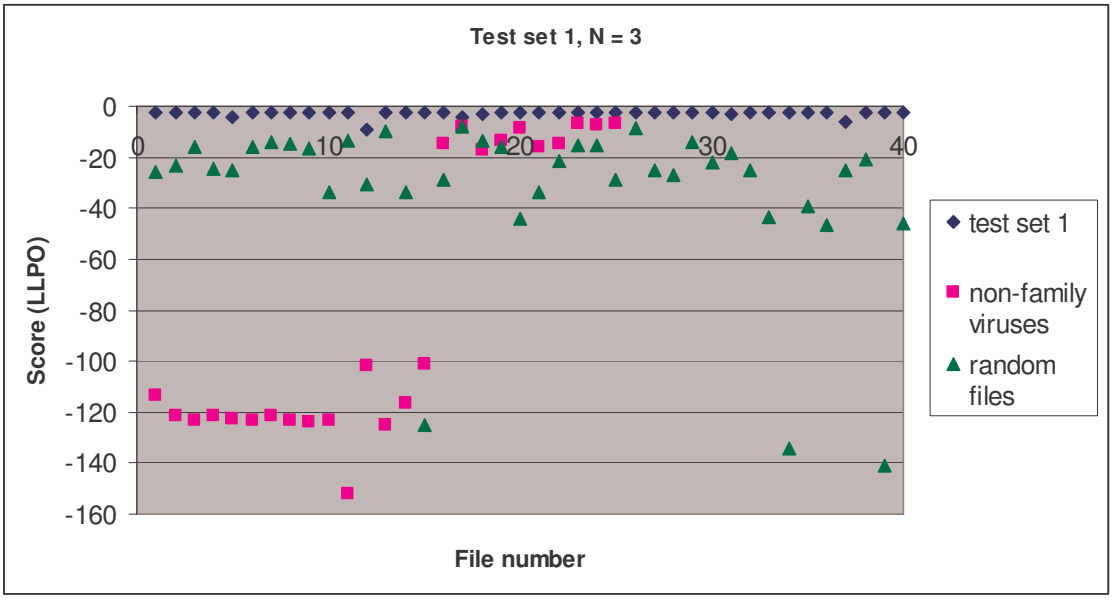
Table A-5 Similarity scores between random “normal” files.

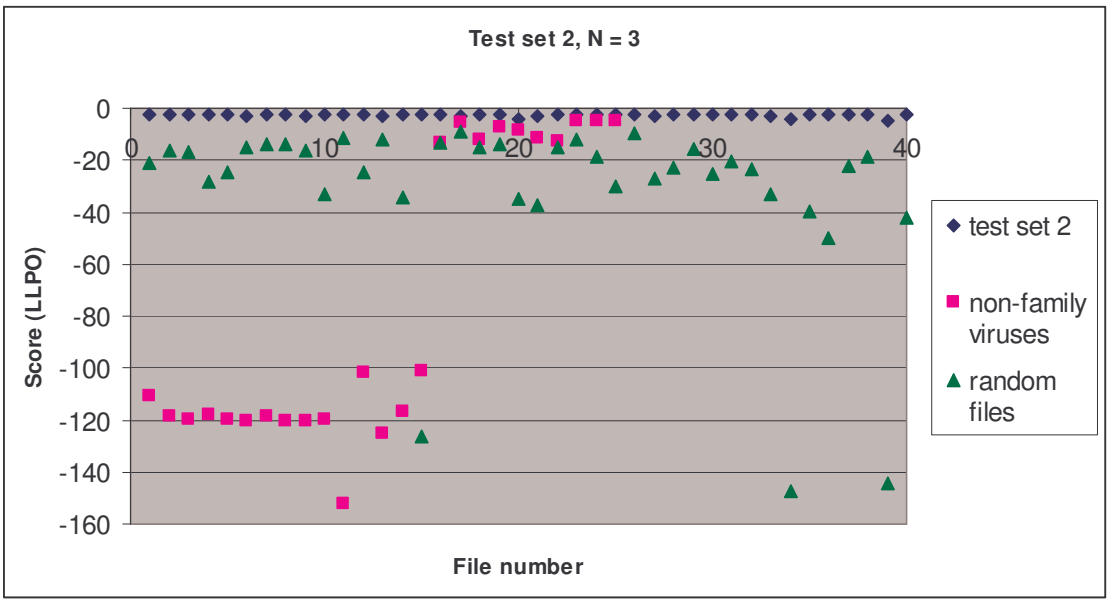
Similarity scores between files:										
IDA_R0	IDA_R1	0.35683	IDA_R3	IDA_R13	0.33470	IDA_R8	IDA_R11	0.18961	min	0.14369
IDA_R0	IDA_R2	0.50040	IDA_R3	IDA_R14	0.23842	IDA_R8	IDA_R12	0.23820	max	0.72535
IDA_R0	IDA_R3	0.33053	IDA_R3	IDA_R15	0.35729	IDA_R8	IDA_R13	0.15451	average	0.36337
IDA_R0	IDA_R4	0.37981	IDA_R3	IDA_R16	0.44687	IDA_R8	IDA_R14	0.14006		
IDA_R0	IDA_R5	0.19924	IDA_R3	IDA_R17	0.37535	IDA_R8	IDA_R15	0.20801		
IDA_R0	IDA_R6	0.19600	IDA_R3	IDA_R18	0.42995	IDA_R8	IDA_R16	0.27208		
IDA_R0	IDA_R7	0.19905	IDA_R3	IDA_R19	0.27338	IDA_R8	IDA_R17	0.22085		
IDA_R0	IDA_R8	0.19984	IDA_R4	IDA_R5	0.18656	IDA_R8	IDA_R18	0.24303		
IDA_R0	IDA_R9	0.33228	IDA_R4	IDA_R6	0.17777	IDA_R8	IDA_R19	0.15206		
IDA_R0	IDA_R10	0.49773	IDA_R4	IDA_R7	0.18059	IDA_R9	IDA_R10	0.49678		
IDA_R0	IDA_R11	0.41739	IDA_R4	IDA_R8	0.18726	IDA_R9	IDA_R11	0.30930		
IDA_R0	IDA_R12	0.38726	IDA_R4	IDA_R9	0.37206	IDA_R9	IDA_R12	0.27024		
IDA_R0	IDA_R13	0.29789	IDA_R4	IDA_R10	0.51310	IDA_R9	IDA_R13	0.34013		
IDA_R0	IDA_R14	0.31944	IDA_R4	IDA_R11	0.34440	IDA_R9	IDA_R14	0.25781		
IDA_R0	IDA_R15	0.46465	IDA_R4	IDA_R12	0.36972	IDA_R9	IDA_R15	0.38430		
IDA_R0	IDA_R16	0.48780	IDA_R4	IDA_R13	0.36090	IDA_R9	IDA_R16	0.44825		
IDA_R0	IDA_R17	0.41608	IDA_R4	IDA_R14	0.25833	IDA_R9	IDA_R17	0.41396		
IDA_R0	IDA_R18	0.39995	IDA_R4	IDA_R15	0.39103	IDA_R9	IDA_R18	0.36174		
IDA_R0	IDA_R19	0.34073	IDA_R4	IDA_R16	0.48730	IDA_R9	IDA_R19	0.28417		
IDA_R1	IDA_R2	0.45579	IDA_R4	IDA_R17	0.42200	IDA_R10	IDA_R11	0.45079		
IDA_R1	IDA_R3	0.29938	IDA_R4	IDA_R18	0.44600	IDA_R10	IDA_R12	0.45866		
IDA_R1	IDA_R4	0.35691	IDA_R4	IDA_R19	0.30770	IDA_R10	IDA_R13	0.44319		
IDA_R1	IDA_R5	0.17400	IDA_R5	IDA_R6	0.89691	IDA_R10	IDA_R14	0.35968		
IDA_R1	IDA_R6	0.17063	IDA_R5	IDA_R7	0.91066	IDA_R10	IDA_R15	0.49985		
IDA_R1	IDA_R7	0.17639	IDA_R5	IDA_R8	0.93395	IDA_R10	IDA_R16	0.65204		
IDA_R1	IDA_R8	0.17465	IDA_R5	IDA_R9	0.16720	IDA_R10	IDA_R17	0.52560		
IDA_R1	IDA_R9	0.24162	IDA_R5	IDA_R10	0.26957	IDA_R10	IDA_R18	0.51452		
IDA_R1	IDA_R10	0.40046	IDA_R5	IDA_R11	0.18895	IDA_R10	IDA_R19	0.40760		
IDA_R1	IDA_R11	0.43216	IDA_R5	IDA_R12	0.23733	IDA_R11	IDA_R12	0.36396		
IDA_R1	IDA_R12	0.67496	IDA_R5	IDA_R13	0.15394	IDA_R11	IDA_R13	0.31181		
IDA_R1	IDA_R13	0.24293	IDA_R5	IDA_R14	0.13945	IDA_R11	IDA_R14	0.29316		
IDA_R1	IDA_R14	0.26337	IDA_R5	IDA_R15	0.20742	IDA_R11	IDA_R15	0.51267		
IDA_R1	IDA_R15	0.45401	IDA_R5	IDA_R16	0.27140	IDA_R11	IDA_R16	0.45261		
IDA_R1	IDA_R16	0.40808	IDA_R5	IDA_R17	0.22024	IDA_R11	IDA_R17	0.36685		
IDA_R1	IDA_R17	0.34480	IDA_R5	IDA_R18	0.24225	IDA_R11	IDA_R18	0.41693		
IDA_R1	IDA_R18	0.41433	IDA_R5	IDA_R19	0.15141	IDA_R11	IDA_R19	0.30487		
IDA_R1	IDA_R19	0.27158	IDA_R6	IDA_R7	0.88308	IDA_R12	IDA_R13	0.27602		
IDA_R2	IDA_R3	0.48679	IDA_R6	IDA_R8	0.89003	IDA_R12	IDA_R14	0.28409		
IDA_R2	IDA_R4	0.54079	IDA_R6	IDA_R9	0.16231	IDA_R12	IDA_R15	0.38460		
IDA_R2	IDA_R5	0.27792	IDA_R6	IDA_R10	0.26633	IDA_R12	IDA_R16	0.45005		
IDA_R2	IDA_R6	0.27305	IDA_R6	IDA_R11	0.18593	IDA_R12	IDA_R17	0.36188		
IDA_R2	IDA_R7	0.27697	IDA_R6	IDA_R12	0.23077	IDA_R12	IDA_R18	0.43837		
IDA_R2	IDA_R8	0.27855	IDA_R6	IDA_R13	0.13848	IDA_R12	IDA_R19	0.30907		
IDA_R2	IDA_R9	0.47721	IDA_R6	IDA_R14	0.13603	IDA_R13	IDA_R14	0.25747		
IDA_R2	IDA_R10	0.72404	IDA_R6	IDA_R15	0.20427	IDA_R13	IDA_R15	0.37897		
IDA_R2	IDA_R11	0.45543	IDA_R6	IDA_R16	0.26421	IDA_R13	IDA_R16	0.41097		
IDA_R2	IDA_R12	0.49804	IDA_R6	IDA_R17	0.20671	IDA_R13	IDA_R17	0.42617		
IDA_R2	IDA_R13	0.47001	IDA_R6	IDA_R18	0.23949	IDA_R13	IDA_R18	0.39149		
IDA_R2	IDA_R14	0.32956	IDA_R6	IDA_R19	0.14545	IDA_R13	IDA_R19	0.27386		
IDA_R2	IDA_R15	0.53073	IDA_R7	IDA_R8	0.90905	IDA_R14	IDA_R15	0.34984		
IDA_R2	IDA_R16	0.72535	IDA_R7	IDA_R9	0.16587	IDA_R14	IDA_R16	0.31725		
IDA_R2	IDA_R17	0.51154	IDA_R7	IDA_R10	0.27080	IDA_R14	IDA_R17	0.32478		
IDA_R2	IDA_R18	0.53837	IDA_R7	IDA_R11	0.18709	IDA_R14	IDA_R18	0.27324		
IDA_R2	IDA_R19	0.40102	IDA_R7	IDA_R12	0.23494	IDA_R14	IDA_R19	0.24026		
IDA_R3	IDA_R4	0.45359	IDA_R7	IDA_R13	0.14106	IDA_R15	IDA_R16	0.54225		
IDA_R3	IDA_R5	0.14913	IDA_R7	IDA_R14	0.13775	IDA_R15	IDA_R17	0.40120		
IDA_R3	IDA_R6	0.14369	IDA_R7	IDA_R15	0.20724	IDA_R15	IDA_R18	0.46115		
IDA_R3	IDA_R7	0.14617	IDA_R7	IDA_R16	0.26824	IDA_R15	IDA_R19	0.36554		
IDA_R3	IDA_R8	0.15209	IDA_R7	IDA_R17	0.20990	IDA_R16	IDA_R17	0.47555		
IDA_R3	IDA_R9	0.32238	IDA_R7	IDA_R18	0.23978	IDA_R16	IDA_R18	0.51024		
IDA_R3	IDA_R10	0.44973	IDA_R7	IDA_R19	0.14865	IDA_R16	IDA_R19	0.36608		
IDA_R3	IDA_R11	0.28466	IDA_R8	IDA_R9	0.16777	IDA_R17	IDA_R18	0.44026		
IDA_R3	IDA_R12	0.31646	IDA_R8	IDA_R10	0.27017	IDA_R17	IDA_R19	0.31786		
						IDA_R18	IDA_R19	0.30629		

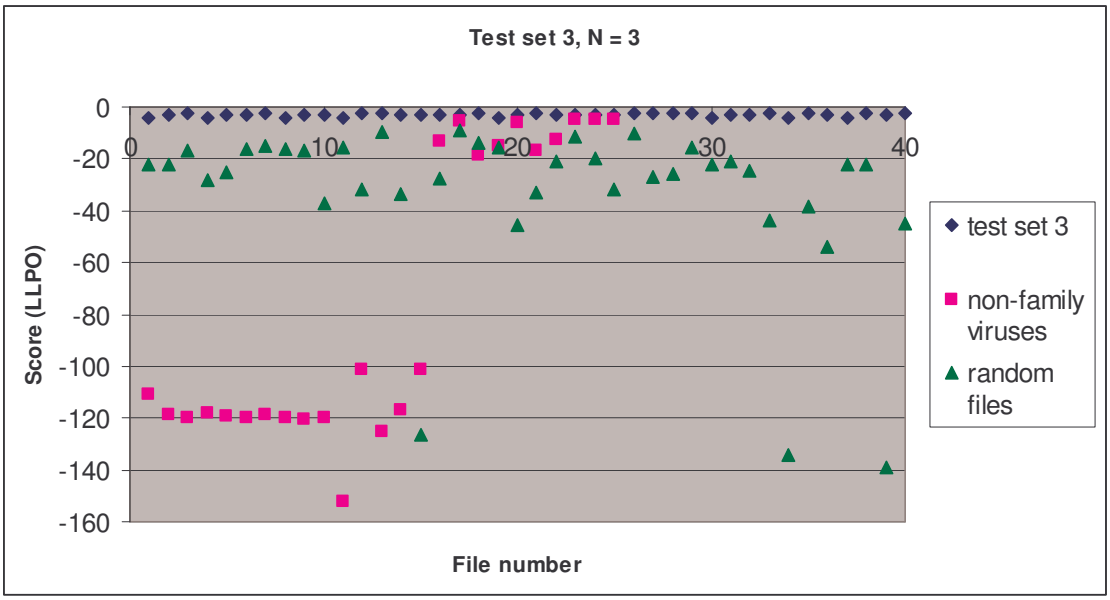
Appendix B: HMM training and testing results

Table B-1 Log likelihood per opcode (LLPO) of family viruses, non-family viruses and random files. The 5 graphs correspond to the 5 models with N = 3.






Test set 2, N = 3
Score (LLPO)
0
-20
-40
-60
-80
-100
-120
-140
-160
0
10
20
30
40
File number
test set 2
non-family viruses
random files


Test set 3, N = 3
Score (LLPO)
0
-20
-40
-60
-80
-100
-120
-140
-160
0
10
20
30
40
File number
test set 3
non-family viruses
random files

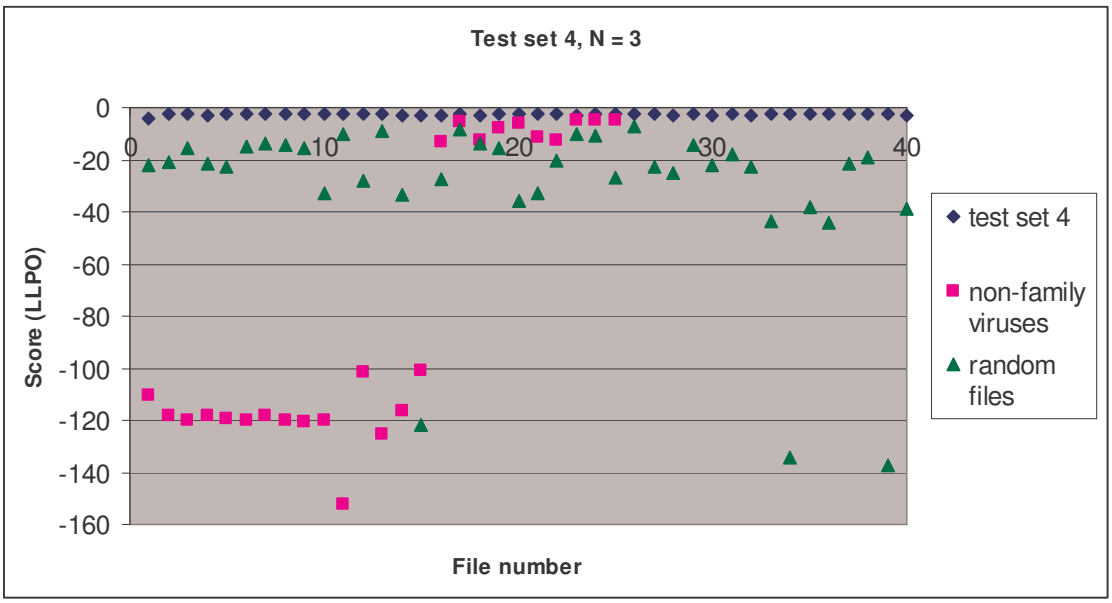
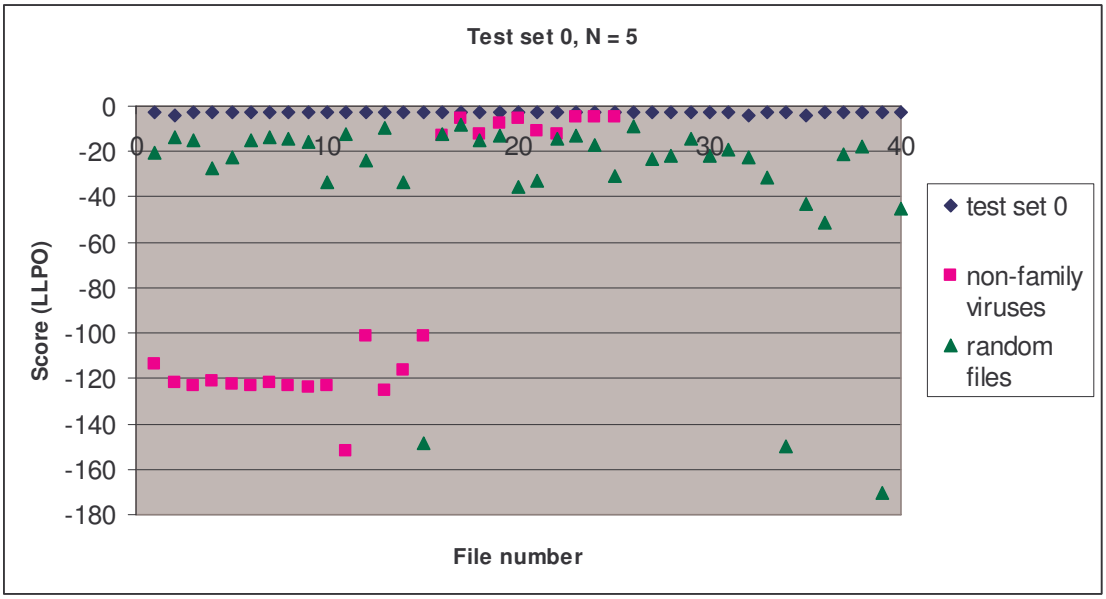
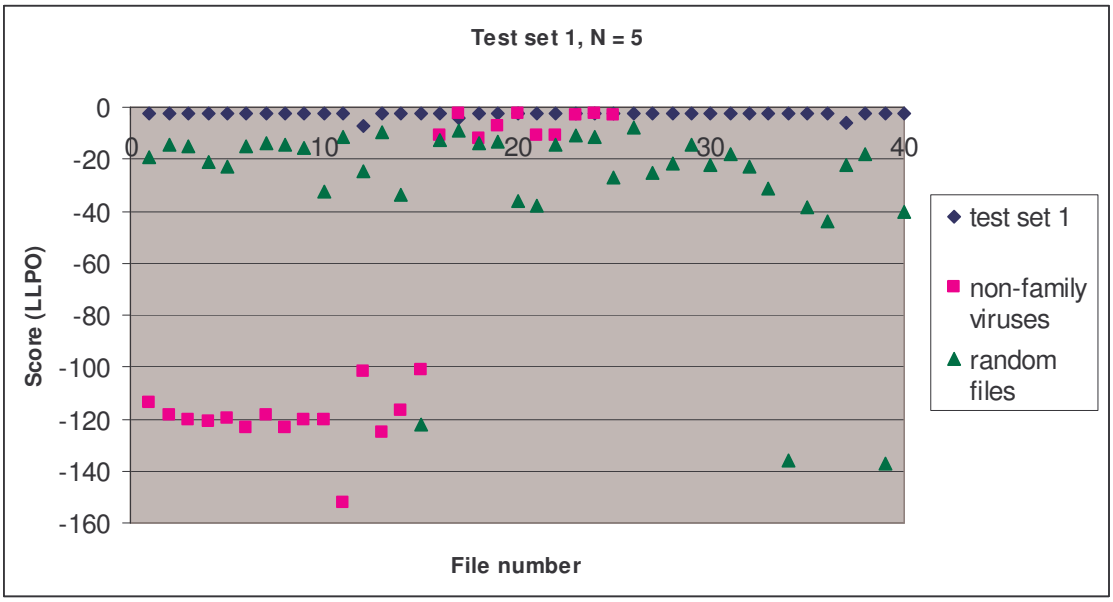
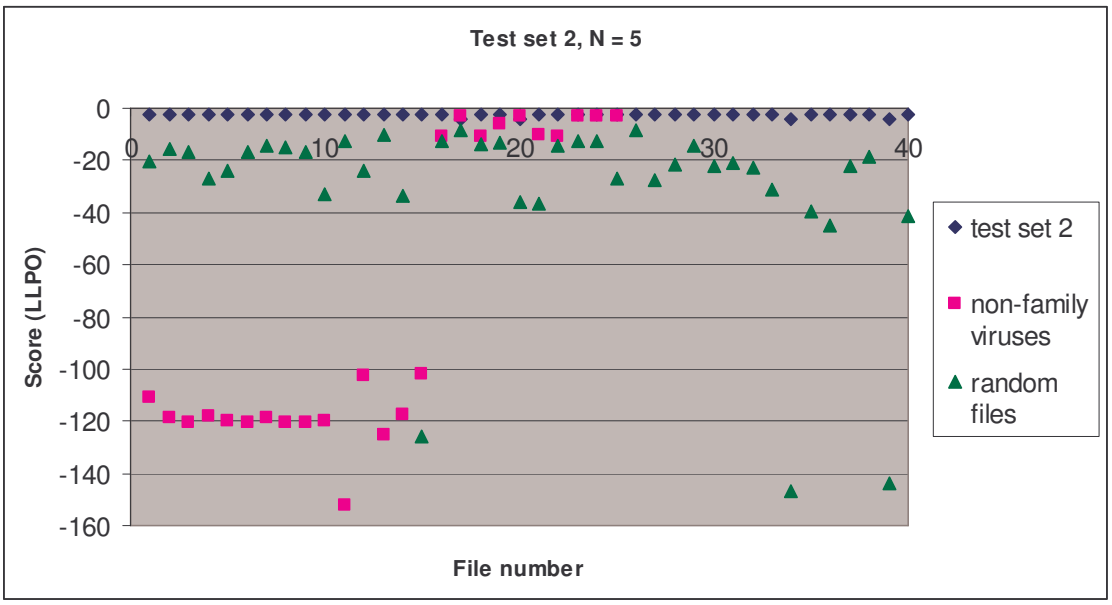
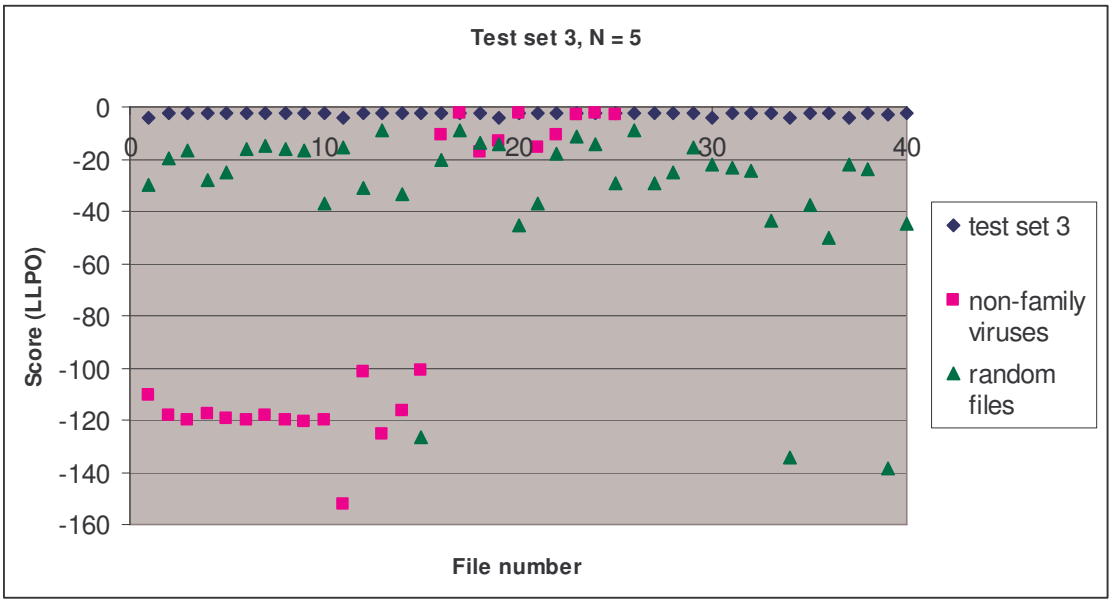

Test set 4, N = 3
0
-20
-40
-60
-80
-100
-120
-140
-160
0
10
20
30
40
Score (LLPO)
File number
test set 4
non-family viruses
random files

Table B-2 Log likelihood per opcode (LLPO) of family viruses, non-family viruses and random files. The 5 graphs correspond to the 5 models with N = 5.






Test set 2, N = 5
Score (LLPO)
0
-20
-40
-60
-80
-100
-120
-140
-160
0
10
20
30
40
File number
test set 2
non-family viruses
random files


Test set 3, N = 5
Score (LLPO)
0
-20
-40
-60
-80
-100
-120
-140
-160
0
10
20
30
40
File number
test set 3
non-family viruses
random files

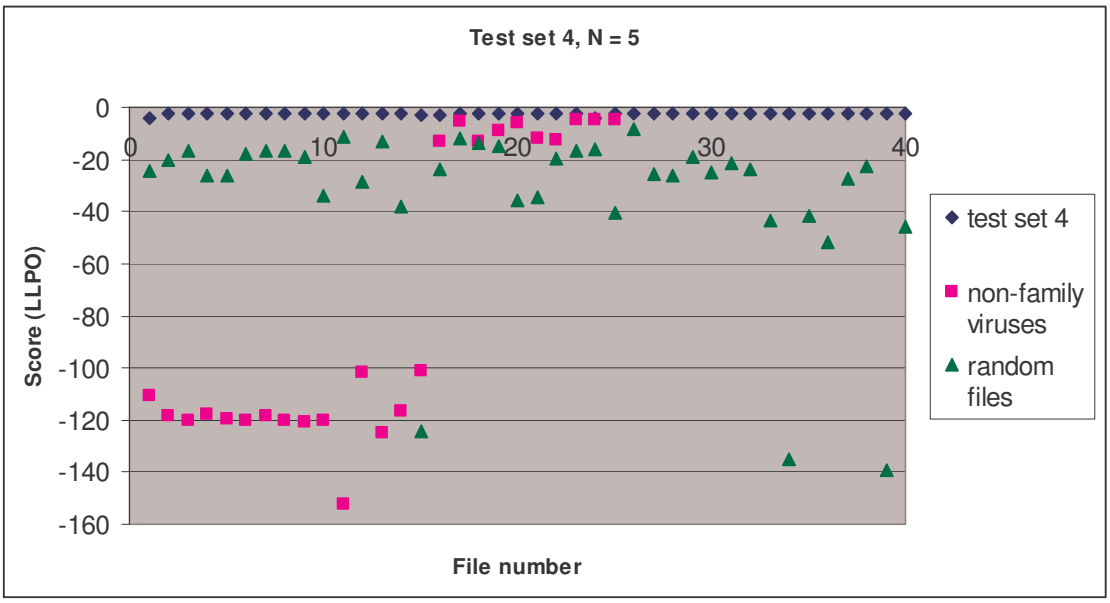

Test set 4, N = 5
0
-20
-40
-60
-80
-100
-120
-140
-160
0
10
20
30
40
Score (LLPO)
File number
test set 4
non-family viruses
random files

Table B-3 Raw LLPO scores of all 105 programs returned by the 25 HMMs. The scores are grouped according to the test set used by an HMM. For each test set, 5 models with N = 2 to 6 were tested.

Test set 0	N = 2	N = 3	N = 4	N = 5	N = 6	Test set 1	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the test set (same family viruses):						Files in the test set (same family viruses):					
IDA_N0	-2.83844	-2.69903	-2.6256	-2.60804	-2.52266	IDA_N40	-2.76366	-2.68011	-2.58964	-2.53576	-2.51455
IDA_N1	-4.38048	-5.85754	-4.19275	-4.1669	-4.06707	IDA_N41	-2.66436	-2.58935	-2.5111	-2.47107	-2.45238
IDA_N2	-2.85605	-2.7188	-2.68132	-2.67629	-2.55774	IDA_N42	-2.69348	-2.61696	-2.52229	-2.4727	-2.4123
IDA_N3	-2.68468	-4.33065	-2.49798	-2.47691	-2.39205	IDA_N43	-2.67667	-2.65022	-2.53879	-2.47592	-2.48122
IDA_N4	-2.78905	-4.34511	-2.58534	-2.55277	-2.47677	IDA_N44	-2.81877	-4.30004	-2.66655	-2.60316	-4.10904
IDA_N5	-2.87672	-4.34558	-2.65451	-2.64242	-2.55389	IDA_N45	-2.71112	-2.64813	-2.57352	-2.51464	-4.11693
IDA_N6	-2.79097	-2.65019	-2.62367	-2.61525	-2.48092	IDA_N46	-2.68092	-2.61321	-2.51652	-2.42621	-2.42194
IDA_N7	-2.692	-4.34712	-2.4885	-2.47446	-2.39059	IDA_N47	-2.69872	-2.61577	-2.52154	-2.45126	-2.42594
IDA_N8	-2.82293	-4.45772	-2.65826	-2.63877	-2.52677	IDA_N48	-2.83159	-2.7425	-2.67465	-2.60111	-2.58626
IDA_N9	-2.71437	-4.45754	-2.52941	-2.51297	-2.42621	IDA_N49	-2.6207	-2.5232	-2.42938	-2.37143	-2.366
IDA_N10	-2.77855	-4.31873	-2.56441	-2.53805	-2.42304	IDA_N50	-2.61617	-2.55355	-2.46196	-2.41996	-2.3982
IDA_N11	-2.68199	-4.44285	-2.47388	-2.44936	-2.35678	IDA_N51	-7.58719	-9.03848	-7.40715	-7.3438	-8.87874
IDA_N12	-2.85616	-2.7279	-2.65932	-2.65164	-2.55999	IDA_N52	-2.64667	-2.5732	-2.47363	-2.4236	-2.41241
IDA_N13	-2.73863	-4.41354	-2.53999	-2.50189	-2.38622	IDA_N53	-2.61651	-2.54794	-2.4426	-2.36823	-2.33674
IDA_N14	-2.77855	-4.29042	-2.57118	-2.55275	-2.45347	IDA_N54	-2.73205	-2.66418	-2.57222	-2.50114	-2.48516
IDA_N15	-2.81468	-4.35899	-2.56416	-2.55566	-2.47357	IDA_N55	-2.73317	-2.64225	-2.55513	-2.47167	-2.4622
IDA_N16	-2.74838	-2.63712	-2.56975	-2.56795	-2.46139	IDA_N56	-4.48225	-4.43088	-4.32367	-4.24991	-5.90228
IDA_N17	-2.76431	-2.62497	-2.58408	-2.56227	-2.46164	IDA_N57	-2.91714	-2.86797	-2.76512	-2.70085	-2.69654
IDA_N18	-2.77806	-4.51851	-2.60557	-2.58518	-2.4591	IDA_N58	-2.70093	-2.63116	-2.54547	-2.48439	-2.46275
IDA_N19	-2.79064	-4.45237	-2.58552	-2.57452	-2.50117	IDA_N59	-2.73989	-2.68482	-2.55744	-2.5066	-2.49595
IDA_N20	-2.82825	-2.68674	-2.65981	-2.64895	-2.48132	IDA_N60	-2.70015	-2.6004	-2.5185	-2.46014	-2.41017
IDA_N21	-2.71906	-2.55134	-2.49386	-2.48976	-2.37984	IDA_N61	-2.6528	-2.60145	-2.51327	-2.4788	-2.43913
IDA_N22	-2.85215	-4.38385	-2.64538	-2.62988	-2.52194	IDA_N62	-2.68543	-2.59348	-2.49426	-2.4262	-2.45459
IDA_N23	-2.79084	-4.44067	-2.5707	-2.54889	-2.46194	IDA_N63	-2.76852	-2.72129	-2.62608	-2.55203	-2.55381
IDA_N24	-2.74196	-2.58964	-2.57811	-2.54555	-2.43279	IDA_N64	-2.65427	-2.57621	-2.48448	-2.41291	-2.43595
IDA_N25	-2.83737	-2.69987	-2.65412	-2.64446	-2.51287	IDA_N65	-2.75828	-2.65424	-2.5927	-2.51481	-2.51811
IDA_N26	-2.75602	-2.59864	-2.5706	-2.55371	-2.45323	IDA_N66	-2.82538	-2.72391	-2.62952	-2.56258	-2.57424
IDA_N27	-2.74015	-4.48543	-2.57684	-2.55921	-2.44652	IDA_N67	-2.78551	-2.70432	-2.61095	-2.55111	-2.5417
IDA_N28	-2.79382	-4.31769	-2.61515	-2.59072	-2.46932	IDA_N68	-2.61916	-2.52407	-2.41714	-2.35491	-2.3518
IDA_N29	-2.81342	-2.65369	-2.59382	-2.58248	-2.46846	IDA_N69	-2.71171	-2.6363	-2.53841	-2.48086	-2.45506
IDA_N30	-2.90366	-2.76041	-2.71772	-2.69408	-2.59481	IDA_N70	-2.78421	-2.78217	-2.63006	-2.57823	-4.15636
IDA_N31	-4.43492	-5.78653	-4.2053	-4.17633	-4.08559	IDA_N71	-2.78259	-2.74365	-2.61111	-2.56922	-2.57302
IDA_N32	-2.78984	-2.65151	-2.60353	-2.57938	-2.51671	IDA_N72	-2.76467	-2.675	-2.58198	-2.51358	-2.50767
IDA_N33	-2.71116	-2.54243	-2.49379	-2.48103	-2.35476	IDA_N73	-2.81895	-2.73003	-2.6272	-2.55822	-2.57578
IDA_N34	-4.40097	-5.77089	-4.1931	-4.183	-4.07268	IDA_N74	-2.73572	-2.64437	-2.54353	-2.48121	-2.46222
IDA_N35	-2.83606	-2.69768	-2.64939	-2.63737	-2.53085	IDA_N75	-2.77316	-2.69325	-2.58006	-2.54218	-2.50283
IDA_N36	-2.80357	-2.6256	-2.59359	-2.56195	-2.45723	IDA_N76	-6.072	-5.9712	-5.92025	-5.83644	-5.85622
IDA_N37	-2.80591	-4.37577	-2.61369	-2.59847	-2.48674	IDA_N77	-2.71058	-2.66658	-2.54104	-2.47233	-2.44444
IDA_N38	-2.93256	-4.36418	-2.75564	-2.74424	-2.6344	IDA_N78	-2.63596	-2.56586	-2.4402	-2.42373	-2.3633
IDA_N39	-2.7216	-4.3746	-2.50902	-2.4919	-2.37628	IDA_N79	-2.80304	-2.7073	-2.60189	-2.55057	-4.15786
min LLPO	-4.43492	-5.85754	-4.2053	-4.183	-4.08559	min LLPO	-7.58719	-9.03848	-7.40715	-7.3438	-8.87874

Test set 0	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the comparison set (other non-family viruses):					
IDA_V0	-110.68	-110.537	-110.51	-113.672	-116.88
IDA_V1	-118.423	-118.391	-118.353	-121.55	-121.655
IDA_V2	-120.033	-119.994	-119.985	-123.12	-126.317
IDA_V3	-118.026	-117.894	-117.861	-121.052	-124.29
IDA_V4	-119.478	-119.441	-119.435	-122.557	-122.634
IDA_V5	-120.159	-120.023	-120.001	-123.151	-126.352
IDA_V6	-118.432	-118.405	-118.378	-121.564	-124.82
IDA_V7	-120.167	-120.034	-120.01	-123.166	-126.374
IDA_V8	-120.567	-120.541	-120.526	-123.673	-126.889
IDA_V9	-120.013	-119.976	-119.965	-123.1	-123.188
IDA_V10	-152.12	-152.01	-152.124	-152.131	-151.978
IDA_V11	-101.51	-101.422	-101.479	-101.473	-101.482
IDA_V12	-125.284	-125.185	-125.229	-125.227	-125.14
IDA_V13	-116.654	-116.559	-116.657	-116.667	-116.685
IDA_V14	-101.059	-100.975	-101.033	-101.03	-101.034
IDA_V15	-11.0989	-12.9583	-12.9958	-12.9572	-12.8483
IDA_V16	-2.99929	-5.40678	-5.37187	-5.33721	-5.30616
IDA_V17	-10.9101	-12.1256	-12.1616	-12.1392	-13.36
IDA_V18	-6.07548	-7.51274	-7.50533	-7.48802	-8.96855
IDA_V19	-2.95565	-5.73581	-5.74302	-5.68135	-5.61087
IDA_V20	-10.1124	-11.1703	-11.21	-11.1938	-12.2449
IDA_V21	-10.8532	-12.6338	-12.6657	-12.6302	-14.45
IDA_V22	-3.06677	-4.89731	-4.8795	-4.86404	-6.75131
IDA_V23	-3.04203	-4.96946	-4.94235	-4.92553	-4.82566
IDA_V24	-3.0499	-4.94289	-4.941	-4.9183	-4.81091
max LLPO	-2.95565	-4.89731	-4.8795	-4.86404	-4.81091
Files in the comparison set (random files):					
IDA_R0	-20.3522	-24.4795	-20.1793	-20.1959	-25.7882
IDA_R1	-13.9877	-24.3116	-14.7271	-13.9233	-24.204
IDA_R2	-14.9357	-16.5983	-14.8663	-14.9016	-15.7212
IDA_R3	-27.6756	-31.0684	-27.647	-27.6873	-31.0792
IDA_R4	-22.7756	-25.8777	-22.7729	-22.8071	-25.8243
IDA_R5	-15.1323	-16.2721	-15.0734	-15.113	-15.5831
IDA_R6	-13.7367	-14.7423	-13.6801	-13.7221	-14.1334
IDA_R7	-14.1954	-15.2444	-14.1447	-14.1902	-14.5943
IDA_R8	-15.8122	-16.9595	-15.7393	-15.7798	-16.2559
IDA_R9	-33.7738	-37.8438	-33.7409	-33.7792	-37.8095
IDA_R10	-12.2689	-17.6877	-12.2309	-12.267	-16.6443
IDA_R11	-23.8743	-30.9366	-23.7247	-23.7407	-30.8355
IDA_R12	-9.48983	-10.4038	-10.3103	-9.50058	-10.3326
IDA_R13	-33.6615	-33.7398	-33.6666	-33.7372	-33.6222
IDA_R14	-148.522	-152.084	-148.487	-148.489	-153.152
IDA_R15	-12.2724	-28.6864	-12.0183	-12.0659	-29.7013
IDA_R16	-8.06632	-8.90711	-7.99737	-8.02997	-8.03371
IDA_R17	-14.7949	-16.0352	-14.7868	-14.8274	-18.3719
IDA_R18	-13.0679	-16.4832	-12.9911	-13.0175	-15.7305
IDA_R19	-35.6981	-46.1659	-35.6743	-35.6973	-46.0777
IDA_R20	-33.1515	-33.8698	-35.4995	-33.0993	-35.4245
IDA_R21	-14.2326	-21.767	-14.113	-14.1297	-21.7071
IDA_R22	-12.9223	-13.8657	-12.8723	-12.895	-14.7197
IDA_R23	-16.9245	-21.3879	-17.77	-16.9387	-21.244
IDA_R24	-30.9469	-32.7376	-30.8959	-30.9188	-33.5087
IDA_R25	-9.16703	-10.7777	-9.04651	-9.05832	-10.2173
IDA_R26	-22.6304	-28.2234	-27.2185	-23.3968	-35.911
IDA_R27	-21.8092	-26.9106	-21.7096	-21.747	-26.8393
IDA_R28	-14.3619	-15.5332	-14.3302	-14.359	-14.2727
IDA_R29	-22.0801	-25.3197	-21.9719	-22.0301	-22.023
IDA_R30	-19.172	-20.1903	-19.1305	-19.1455	-21.1287
IDA_R31	-22.5469	-24.8483	-22.5491	-22.5927	-24.7886
IDA_R32	-31.503	-43.6435	-31.2799	-31.3329	-43.5575
IDA_R33	-149.001	-149.753	-149.077	-149.735	-149.66
IDA_R34	-42.8888	-43.5834	-42.7889	-42.8023	-43.542
IDA_R35	-51.267	-54.4469	-51.2	-51.2107	-54.3881
IDA_R36	-21.458	-21.5564	-21.4072	-21.4287	-24.0999
IDA_R37	-17.9681	-21.4674	-17.7994	-17.8171	-23.2231
IDA_R38	-169.192	-169.933	-169.2	-170.533	-171.988
IDA_R39	-45.4978	-49.2993	-45.4443	-45.4541	-52.5257
max LLPO	-8.06632	-8.90711	-7.99737	-8.02997	-8.03371

Test set 1	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the comparison set (other non-family viruses):					
IDA_V0	-110.536	-113.711	-110.501	-113.692	-116.841
IDA_V1	-118.371	-121.604	-118.343	-118.388	-124.765
IDA_V2	-119.985	-123.157	-119.95	-120.005	-126.302
IDA_V3	-117.886	-121.094	-117.853	-121.068	-124.252
IDA_V4	-119.443	-122.598	-119.408	-119.468	-125.714
IDA_V5	-120.012	-123.186	-119.98	-123.182	-126.339
IDA_V6	-118.4	-121.625	-118.366	-118.397	-124.793
IDA_V7	-120.036	-123.211	-120	-123.186	-126.336
IDA_V8	-120.521	-123.716	-120.49	-120.543	-126.868
IDA_V9	-119.976	-123.149	-119.941	-120.001	-126.283
IDA_V10	-152.129	-152.073	-152.036	-151.955	-152.017
IDA_V11	-101.476	-101.458	-101.353	-101.4	-101.429
IDA_V12	-125.255	-125.203	-125.129	-125.125	-125.186
IDA_V13	-116.641	-116.635	-116.55	-116.564	-116.599
IDA_V14	-101.031	-101.01	-100.905	-100.953	-100.978
IDA_V15	-11.0055	-15.0036	-12.8509	-10.8721	-14.8833
IDA_V16	-2.86441	-7.99111	-5.24582	-2.67224	-7.76235
IDA_V17	-12.1081	-17.4324	-13.2475	-12.0539	-17.2344
IDA_V18	-7.45631	-13.6628	-8.80031	-7.3797	-13.3882
IDA_V19	-2.82419	-8.7245	-5.60009	-2.6033	-8.64268
IDA_V20	-11.1813	-15.9209	-10.9923	-11.0869	-14.5662
IDA_V21	-10.7554	-14.6156	-12.5222	-10.6257	-14.4848
IDA_V22	-2.94324	-6.93559	-2.72727	-2.72782	-4.72337
IDA_V23	-2.91692	-7.09299	-2.70189	-2.68671	-4.77661
IDA_V24	-2.93749	-7.02483	-2.7352	-2.70715	-4.78438
max LLPO	-2.82419	-6.93559	-2.70189	-2.6033	-4.72337
Files in the comparison set (random files):					
IDA_R0	-19.0813	-25.6181	-25.4629	-19.3923	-30.964
IDA_R1	-12.9742	-23.3083	-25.8326	-14.1531	-33.5984
IDA_R2	-14.892	-15.7651	-17.3202	-14.9418	-18.2097
IDA_R3	-20.9218	-24.3491	-27.5827	-21.1412	-32.0876
IDA_R4	-22.7361	-25.2426	-26.9897	-22.9385	-32.4967
IDA_R5	-15.0357	-15.6858	-18.4092	-15.1611	-22.9879
IDA_R6	-13.6455	-14.2019	-17.131	-13.7405	-21.7219
IDA_R7	-14.1103	-14.6935	-17.2122	-14.7183	-21.4581
IDA_R8	-15.7075	-16.3599	-19.681	-15.8273	-24.2979
IDA_R9	-32.1023	-33.8338	-36.1359	-32.3185	-45.0598
IDA_R10	-10.0797	-13.3747	-14.3734	-11.3154	-19.7933
IDA_R11	-23.7319	-30.9349	-30.8593	-24.5349	-37.361
IDA_R12	-8.59588	-9.53152	-17.9652	-9.67963	-18.9
IDA_R13	-33.5891	-33.6865	-38.6692	-33.8411	-38.7489
IDA_R14	-120.268	-124.967	-125.499	-122.262	-137.027
IDA_R15	-11.9631	-28.6641	-26.4973	-12.3387	-38.9709
IDA_R16	-8.01384	-8.08398	-14.426	-8.97044	-16.1008
IDA_R17	-13.5173	-13.5915	-15.8967	-13.6832	-20.7959
IDA_R18	-12.9705	-15.7858	-16.37	-13.0381	-18.5487
IDA_R19	-34.6171	-44.0611	-36.6797	-35.8427	-46.6268
IDA_R20	-33.0387	-33.851	-36.1835	-37.9465	-37.0886
IDA_R21	-14.0953	-21.7596	-20.9419	-14.2059	-28.6001
IDA_R22	-9.95689	-15.2951	-21.6112	-10.9678	-31.1503
IDA_R23	-10.6694	-15.2433	-17.7176	-11.7232	-22.1688
IDA_R24	-26.6315	-28.5351	-42.8449	-27.3516	-47.9732
IDA_R25	-7.38424	-8.57564	-9.57934	-7.98664	-11.9263
IDA_R26	-19.4715	-25.3171	-30.06	-25.4243	-34.9636
IDA_R27	-21.694	-26.8971	-28.5276	-21.8715	-34.6
IDA_R28	-14.2948	-14.356	-21.1883	-14.3848	-23.5694
IDA_R29	-21.9533	-22.0916	-28.2818	-22.1719	-28.228
IDA_R30	-18.087	-18.151	-24.0749	-18.2444	-25.1344
IDA_R31	-22.5222	-24.8293	-24.7074	-22.6012	-29.2088
IDA_R32	-31.215	-43.6288	-47.021	-31.3551	-48.8613
IDA_R33	-134.309	-134.301	-135.629	-135.861	-135.683
IDA_R34	-37.7545	-39.0629	-42.0903	-38.5235	-45.3049
IDA_R35	-43.5655	-46.8861	-57.5935	-43.7333	-61.4655
IDA_R36	-21.3869	-24.9779	-29.0406	-22.4636	-40.1465
IDA_R37	-17.8202	-20.6498	-23.1537	-18.2312	-33.7685
IDA_R38	-136.402	-141.157	-140.4	-137.329	-146.51
IDA_R39	-38.6277	-45.6849	-51.1963	-40.0192	-62.7995
max LLPO	-7.38424	-8.08398	-9.57934	-7.98664	-11.9263

Test set 2	N = 2	N = 3	N = 4	N = 5	N = 6	Test set 3	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the test set (same family viruses):						Files in the test set (same family viruses):					
IDA_N80	-2.78596	-2.67798	-2.59714	-2.54315	-2.50417	IDA_N120	-4.40838	-4.3549	-4.24551	-4.21035	-4.16695
IDA_N81	-2.71582	-2.61435	-2.51226	-2.483	-2.44984	IDA_N121	-2.79207	-2.762	-2.64508	-2.62348	-2.58086
IDA_N82	-2.74543	-2.65419	-2.56718	-2.51449	-2.51249	IDA_N122	-2.75272	-2.66651	-2.57969	-2.53086	-2.56025
IDA_N83	-2.78747	-2.70289	-2.61369	-2.55434	-2.53922	IDA_N123	-2.80885	-4.16994	-2.64693	-2.58321	-2.56961
IDA_N84	-2.74214	-2.64291	-2.54089	-2.5011	-2.46332	IDA_N124	-2.79929	-2.72876	-2.61708	-2.56786	-2.53785
IDA_N85	-2.83384	-2.75978	-2.65446	-2.57352	-2.56557	IDA_N125	-2.84085	-2.71591	-2.64955	-2.60904	-2.57639
IDA_N86	-2.76724	-2.6864	-2.56902	-2.49985	-2.47487	IDA_N126	-2.71159	-2.64168	-2.54382	-2.48536	-2.48441
IDA_N87	-2.74763	-2.65757	-2.54139	-2.47209	-4.01189	IDA_N127	-2.75706	-4.32286	-2.62511	-2.579	-2.55852
IDA_N88	-2.78115	-2.70848	-2.59279	-2.51722	-2.48952	IDA_N128	-2.75656	-2.71333	-2.58294	-2.5476	-2.52283
IDA_N89	-2.80629	-2.70219	-2.60907	-2.57565	-2.56103	IDA_N129	-2.80964	-2.81176	-2.65443	-2.63213	-2.62324
IDA_N90	-2.70537	-2.63583	-2.51728	-2.43874	-2.41315	IDA_N130	-4.38085	-4.26365	-4.20121	-4.1437	-4.11532
IDA_N91	-2.72608	-2.6466	-2.50874	-2.46342	-4.22017	IDA_N131	-2.68634	-2.61763	-2.5276	-2.50032	-2.44971
IDA_N92	-2.81399	-2.72244	-2.64023	-2.60376	-2.54441	IDA_N132	-2.7368	-2.64647	-2.55378	-2.48693	-2.45418
IDA_N93	-2.767	-2.70535	-2.57776	-2.54274	-2.49414	IDA_N133	-2.80202	-2.70288	-2.63477	-2.59805	-2.56288
IDA_N94	-2.7922	-2.70282	-2.5933	-2.53294	-2.51457	IDA_N134	-2.76731	-2.6989	-2.58557	-2.5438	-2.508
IDA_N95	-2.75955	-2.65987	-2.55261	-2.48986	-2.49216	IDA_N135	-2.80256	-2.70427	-2.64668	-2.60969	-2.54873
IDA_N96	-2.79448	-2.73039	-2.60746	-4.19023	-2.49159	IDA_N136	-2.76941	-2.73932	-2.6191	-2.59471	-2.5523
IDA_N97	-2.70511	-2.61656	-2.50253	-2.45814	-2.42864	IDA_N137	-2.70422	-2.65856	-2.54594	-2.51609	-2.47892
IDA_N98	-2.70815	-2.64074	-2.5102	-2.45903	-2.41277	IDA_N138	-4.29175	-4.22611	-4.109	-4.04495	-4.05201
IDA_N99	-4.34287	-4.25866	-4.14343	-4.07437	-4.07465	IDA_N139	-2.7641	-2.70675	-2.58996	-2.55617	-2.54058
IDA_N100	-2.85729	-2.74847	-2.65396	-2.59004	-2.60693	IDA_N140	-2.75294	-2.65459	-2.56703	-2.53027	-2.48005
IDA_N101	-2.78114	-2.69631	-2.58819	-2.52942	-2.50849	IDA_N141	-2.84668	-2.80375	-2.70096	-2.64429	-2.64691
IDA_N102	-2.76594	-2.66987	-2.55994	-2.5083	-4.03463	IDA_N142	-2.80492	-2.74301	-2.64392	-2.60063	-2.59846
IDA_N103	-2.74484	-2.66455	-2.55925	-2.49072	-2.47662	IDA_N143	-2.81709	-2.75421	-2.62445	-2.58204	-2.54805
IDA_N104	-2.70546	-2.59114	-2.50912	-2.44322	-2.40703	IDA_N144	-2.81491	-2.75971	-2.66119	-2.6216	-2.58588
IDA_N105	-2.75187	-2.65959	-2.55598	-2.49245	-2.46596	IDA_N145	-2.76155	-2.66068	-2.59429	-2.53725	-2.52912
IDA_N106	-2.88066	-2.80588	-2.70703	-2.69344	-2.66017	IDA_N146	-2.6636	-2.55819	-2.47953	-2.44591	-2.40288
IDA_N107	-2.78407	-2.69533	-2.59493	-2.53562	-2.51467	IDA_N147	-2.75001	-2.68399	-2.57598	-2.52253	-2.50413
IDA_N108	-2.73623	-2.6356	-2.53705	-2.49401	-2.47551	IDA_N148	-2.63723	-2.59727	-2.4899	-2.45315	-2.43717
IDA_N109	-2.78223	-2.65009	-2.54986	-2.48129	-2.46029	IDA_N149	-4.49808	-4.3908	-4.30824	-4.24797	-4.22151
IDA_N110	-2.80412	-2.69219	-2.58141	-2.51092	-2.4816	IDA_N150	-2.83201	-2.7626	-2.64384	-2.62516	-2.5844
IDA_N111	-2.74461	-2.6614	-2.55099	-2.49983	-4.19169	IDA_N151	-2.78473	-2.73271	-2.59089	-2.55756	-2.54645
IDA_N112	-2.81762	-2.75437	-2.62037	-2.55823	-2.52904	IDA_N152	-2.72347	-2.61939	-2.52003	-2.50119	-2.45198
IDA_N113	-4.53895	-4.46736	-4.35621	-4.37182	-4.28506	IDA_N153	-4.34245	-4.2674	-4.18126	-4.12804	-4.11664
IDA_N114	-2.74666	-2.6584	-2.5499	-2.49236	-2.46588	IDA_N154	-2.68819	-2.62319	-2.51985	-2.48696	-2.45975
IDA_N115	-2.77698	-2.67656	-2.54838	-2.46894	-2.45529	IDA_N155	-2.76686	-2.70078	-2.59012	-2.53217	-2.49779
IDA_N116	-2.78568	-2.66194	-2.52794	-2.4681	-2.45549	IDA_N156	-4.38759	-4.34126	-4.24587	-4.20696	-4.1804
IDA_N117	-2.74814	-2.66958	-2.56053	-2.50569	-2.4633	IDA_N157	-2.70717	-2.64597	-2.53334	-2.48732	-2.44546
IDA_N118	-4.68817	-4.61851	-4.48343	-4.41854	-4.38075	IDA_N158	-2.88093	-2.78789	-2.71011	-2.68702	-2.65899
IDA_N119	-2.7264	-2.6377	-2.52504	-2.45912	-2.43363	IDA_N159	-2.67346	-2.62042	-2.52023	-2.48502	-2.44617
min LLPO	-4.68817	-4.61851	-4.48343	-4.41854	-4.38075	min LLPO	-4.49808	-4.3908	-4.30824	-4.24797	-4.22151

Test set 2	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the comparison set (other non-family viruses):					
IDA_V0	-110.548	-110.572	-110.498	-110.693	-110.664
IDA_V1	-118.396	-118.378	-118.339	-118.538	-118.509
IDA_V2	-120	-119.999	-119.949	-120.152	-120.127
IDA_V3	-117.902	-117.923	-117.849	-118.043	-118.009
IDA_V4	-119.452	-119.46	-119.406	-119.606	-119.602
IDA_V5	-120.027	-120.061	-119.977	-120.178	-120.16
IDA_V6	-118.412	-118.407	-118.359	-118.559	-118.529
IDA_V7	-120.04	-120.068	-119.99	-120.193	-120.163
IDA_V8	-120.542	-120.541	-120.489	-120.689	-120.673
IDA_V9	-119.985	-119.992	-119.938	-120.143	-120.127
IDA_V10	-152.129	-152.17	-152.002	-152.011	-152.068
IDA_V11	-101.482	-101.538	-101.353	-102.176	-104.647
IDA_V12	-125.247	-125.321	-125.12	-125.278	-125.208
IDA_V13	-116.651	-116.679	-116.536	-117.558	-121.365
IDA_V14	-101.034	-101.09	-100.906	-101.725	-104.184
IDA_V15	-11.0526	-12.9766	-10.8516	-10.9109	-12.9371
IDA_V16	-2.91352	-5.37428	-2.71082	-2.87116	-2.83498
IDA_V17	-10.8853	-12.0943	-13.2599	-10.789	-13.384
IDA_V18	-6.03072	-7.44328	-8.81842	-6.00191	-7.45941
IDA_V19	-2.8765	-8.64549	-2.66718	-2.71829	-2.7155
IDA_V20	-10.0731	-11.1388	-12.1581	-9.96582	-12.2788
IDA_V21	-10.8112	-12.6565	-10.5948	-10.6503	-12.6381
IDA_V22	-2.96596	-4.86855	-2.7186	-2.88576	-2.83208
IDA_V23	-2.93546	-4.92886	-2.69302	-2.86526	-2.82199
IDA_V24	-2.94892	-4.91569	-2.71813	-2.88978	-2.79682
max LLPO	-2.8765	-4.86855	-2.66718	-2.71829	-2.7155
Files in the comparison set (random files):					
IDA_R0	-20.241	-21.3017	-32.0329	-20.2415	-27.7415
IDA_R1	-15.5845	-16.4398	-24.1227	-15.7917	-16.6454
IDA_R2	-15.7218	-16.631	-17.3306	-16.5678	-16.5666
IDA_R3	-25.445	-28.2337	-34.3191	-27.2103	-26.2244
IDA_R4	-24.0234	-24.6085	-27.6055	-24.0773	-25.2894
IDA_R5	-15.1889	-15.0671	-18.4239	-16.7348	-15.607
IDA_R6	-13.7968	-13.6796	-17.144	-14.6289	-14.1261
IDA_R7	-14.2576	-14.1326	-17.2242	-15.1259	-15.1338
IDA_R8	-15.8568	-16.3115	-19.6964	-16.8518	-16.2749
IDA_R9	-32.1813	-32.9641	-36.1879	-33.0407	-36.2825
IDA_R10	-10.0854	-11.2118	-13.3016	-12.3539	-13.4084
IDA_R11	-23.8302	-24.4891	-29.4495	-23.8197	-26.592
IDA_R12	-10.3428	-12.0377	-15.4153	-10.439	-10.37
IDA_R13	-33.6998	-34.4903	-37.8368	-33.7281	-33.7347
IDA_R14	-123.763	-126.04	-129.534	-125.611	-138.833
IDA_R15	-12.0766	-13.0947	-26.516	-12.3021	-13.2413
IDA_R16	-8.06468	-8.89833	-11.2414	-8.13268	-8.87471
IDA_R17	-13.5885	-14.8085	-14.7247	-13.6691	-14.8508
IDA_R18	-13.034	-13.7234	-14.9996	-13.057	-13.0304
IDA_R19	-34.6448	-35.1749	-37.7394	-35.7471	-41.4484
IDA_R20	-36.2998	-37.0972	-41.8702	-36.2976	-38.0069
IDA_R21	-14.1682	-14.918	-20.1976	-14.2488	-14.9229
IDA_R22	-10.7376	-12.2021	-21.6265	-12.3659	-11.6717
IDA_R23	-12.5025	-18.6126	-21.1913	-12.5908	-12.5571
IDA_R24	-26.7399	-30.069	-44.5473	-26.987	-27.7515
IDA_R25	-7.91717	-9.69811	-11.2641	-8.51496	-8.5476
IDA_R26	-22.3951	-27.2093	-30.0525	-27.3349	-26.4668
IDA_R27	-21.7747	-22.588	-26.8228	-21.8227	-21.7678
IDA_R28	-14.3795	-15.5379	-20.0434	-14.3344	-14.3286
IDA_R29	-21.9617	-25.1856	-28.24	-22.0525	-22.0477
IDA_R30	-19.1733	-20.1766	-25.1156	-21.2588	-22.1643
IDA_R31	-22.5614	-23.7033	-24.7076	-22.6304	-22.5915
IDA_R32	-31.2966	-33.0363	-45.2626	-31.3909	-31.3026
IDA_R33	-146.932	-147.626	-148.213	-146.924	-148.941
IDA_R34	-39.0954	-39.7153	-44.6395	-39.6642	-41.6695
IDA_R35	-44.8848	-49.9879	-54.4208	-44.9983	-49.3819
IDA_R36	-21.4809	-22.3114	-29.0516	-22.4039	-22.3303
IDA_R37	-17.895	-18.7717	-26.7078	-18.8006	-23.2127
IDA_R38	-143.785	-144.475	-147.741	-143.862	-154.468
IDA_R39	-40.8089	-41.8073	-55.3838	-41.4848	-48.2084
max	-7.91717	-8.89833	-11.2414	-8.13268	-8.5476

Test set 3	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the comparison set (other non-family viruses):					
IDA_V0	-110.536	-110.608	-110.5	-110.525	-110.666
IDA_V1	-118.372	-118.458	-118.343	-118.374	-118.458
IDA_V2	-119.985	-120.053	-119.95	-119.988	-120.137
IDA_V3	-117.885	-117.962	-117.852	-117.874	-118.021
IDA_V4	-119.442	-119.508	-119.407	-119.441	-119.528
IDA_V5	-120.011	-120.082	-119.979	-120.004	-120.171
IDA_V6	-118.399	-118.479	-118.365	-118.381	-118.534
IDA_V7	-120.035	-120.106	-119.999	-120.011	-120.161
IDA_V8	-120.521	-120.598	-120.489	-120.521	-120.681
IDA_V9	-119.976	-120.046	-119.94	-119.978	-120.051
IDA_V10	-152.125	-152.07	-152.031	-152.024	-152.017
IDA_V11	-101.479	-101.464	-101.357	-101.395	-101.446
IDA_V12	-125.252	-125.201	-125.127	-125.154	-125.194
IDA_V13	-116.646	-116.644	-116.556	-116.572	-116.608
IDA_V14	-101.034	-101.017	-100.91	-100.952	-100.993
IDA_V15	-11.0142	-13.049	-12.8581	-10.8223	-12.9563
IDA_V16	-2.86898	-5.50177	-5.24964	-2.67762	-2.78742
IDA_V17	-17.2615	-18.6015	-18.393	-17.0937	-18.5623
IDA_V18	-13.4335	-15.0164	-14.7689	-13.268	-13.431
IDA_V19	-2.82661	-5.84812	-5.60036	-2.59589	-8.60145
IDA_V20	-15.7986	-16.9686	-15.6022	-15.5932	-16.8943
IDA_V21	-10.7661	-12.729	-12.5327	-10.5691	-10.7072
IDA_V22	-2.94866	-4.97958	-2.73242	-2.69981	-2.76739
IDA_V23	-2.92116	-5.04747	-2.70515	-2.67217	-2.73037
IDA_V24	-2.94101	-5.01374	-2.73714	-2.69121	-4.80919
max LLPO	-2.82661	-4.97958	-2.70515	-2.59589	-2.73037
Files in the comparison set (random files):					
IDA_R0	-19.0733	-22.3797	-25.4575	-29.8821	-24.5125
IDA_R1	-14.6656	-22.4553	-27.5165	-19.9896	-26.0485
IDA_R2	-16.5353	-16.5734	-18.1385	-16.5177	-17.3848
IDA_R3	-25.3828	-28.2235	-31.4691	-28.2749	-26.5838
IDA_R4	-25.1484	-25.2332	-29.9919	-25.2205	-27.6808
IDA_R5	-16.1641	-16.232	-20.6588	-16.166	-19.0566
IDA_R6	-14.6479	-14.687	-19.1296	-14.6336	-17.7192
IDA_R7	-16.1979	-16.2406	-20.3352	-16.1853	-18.8588
IDA_R8	-16.846	-16.9113	-21.9509	-16.8412	-20.3313
IDA_R9	-36.9121	-37.0179	-40.1403	-37.0123	-37.8589
IDA_R10	-15.443	-15.4953	-18.6434	-15.5129	-15.5375
IDA_R11	-27.3056	-31.6491	-34.4136	-30.909	-33.1087
IDA_R12	-8.5878	-9.52869	-17.9569	-8.68176	-12.1838
IDA_R13	-33.5869	-33.6842	-38.6606	-33.6463	-37.9048
IDA_R14	-124.857	-126.663	-129.497	-126.715	-128.449
IDA_R15	-11.9634	-27.6113	-26.4995	-20.4132	-30.7991
IDA_R16	-8.80385	-8.86505	-15.2244	-8.88974	-12.1596
IDA_R17	-13.5114	-13.5997	-15.8947	-13.5916	-13.6148
IDA_R18	-12.9635	-15.7827	-17.0633	-14.3712	-16.4929
IDA_R19	-45.607	-45.6405	-47.6154	-45.578	-46.1171
IDA_R20	-33.029	-33.0575	-36.1755	-37.0828	-37.9303
IDA_R21	-14.8482	-21.0235	-22.4521	-17.9806	-23.3207
IDA_R22	-10.6733	-11.661	-20.8805	-11.562	-15.3511
IDA_R23	-14.1798	-19.5891	-23.8061	-14.2806	-16.1485
IDA_R24	-29.152	-31.8685	-44.5021	-29.4037	-39.6263
IDA_R25	-9.06776	-10.2353	-11.8215	-9.10282	-9.17058
IDA_R26	-20.4191	-27.2461	-31.0077	-29.2257	-31.2292
IDA_R27	-22.5432	-26.0522	-29.3701	-25.1704	-28.652
IDA_R28	-15.4499	-15.5185	-21.1844	-15.4242	-18.9564
IDA_R29	-21.9482	-22.1274	-28.2699	-21.946	-25.0892
IDA_R30	-21.1125	-21.1409	-24.07	-23.0999	-23.1083
IDA_R31	-24.7234	-24.8329	-25.7976	-24.7581	-24.81
IDA_R32	-31.2152	-43.6456	-47.017	-43.608	-45.4016
IDA_R33	-134.306	-134.307	-135.627	-134.348	-134.991
IDA_R34	-37.7493	-38.4199	-42.0894	-37.7209	-41.5299
IDA_R35	-49.2985	-53.8402	-61.3921	-50.0184	-50.1512
IDA_R36	-22.222	-22.4281	-30.7306	-22.343	-27.5638
IDA_R37	-22.2145	-22.3416	-27.5314	-24.056	-26.7257
IDA_R38	-138.396	-139.163	-141.724	-138.476	-140.542
IDA_R39	-44.3496	-45.1145	-54.3106	-44.524	-47.8366
max LLPO	-8.5878	-8.86505	-11.8215	-8.68176	-9.17058

Test set 4	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the test set (same family viruses):					
IDA_N160	-4.39243	-4.25642	-4.15688	-4.12541	-4.08538
IDA_N161	-2.77908	-2.64894	-2.54489	-2.51143	-2.47611
IDA_N162	-2.75727	-2.62508	-2.54452	-2.5253	-2.47465
IDA_N163	-2.85864	-2.7459	-2.62878	-2.59791	-2.56974
IDA_N164	-2.79247	-2.65693	-2.57332	-2.53272	-2.49402
IDA_N165	-2.68258	-2.54961	-2.42946	-2.39459	-2.359
IDA_N166	-2.77237	-2.62415	-2.55595	-2.51392	-2.49115
IDA_N167	-2.74633	-2.61362	-2.48832	-2.45684	-2.41554
IDA_N168	-2.83164	-2.68613	-2.61553	-2.57382	-2.55556
IDA_N169	-2.75223	-2.60303	-2.50869	-2.46421	-2.4381
IDA_N170	-2.80635	-2.68039	-2.5928	-2.55065	-2.49167
IDA_N171	-2.79455	-2.66581	-2.58196	-2.53781	-2.49518
IDA_N172	-2.77357	-2.65464	-2.55351	-2.52545	-2.47973
IDA_N173	-2.85727	-2.70765	-2.60495	-2.56565	-2.53795
IDA_N174	-2.93994	-2.84019	-2.75818	-2.74438	-2.7188
IDA_N175	-2.93905	-2.81191	-2.71527	-2.6864	-2.66831
IDA_N176	-2.79106	-2.67423	-2.62019	-2.57352	-2.52349
IDA_N177	-2.87316	-2.72633	-2.62461	-2.56762	-2.51474
IDA_N178	-2.77296	-2.63028	-2.55853	-2.5241	-2.49757
IDA_N179	-2.80715	-2.67119	-2.56987	-2.51599	-2.49416
IDA_N180	-2.75548	-2.61047	-2.50619	-2.45819	-2.43056
IDA_N181	-2.80222	-2.65451	-2.55	-2.51199	-2.45073
IDA_N182	-2.84607	-2.71719	-2.63298	-2.59614	-2.56706
IDA_N183	-2.72344	-2.61417	-4.24962	-4.22606	-4.18224
IDA_N184	-2.773	-2.64818	-2.52487	-2.48326	-2.44597
IDA_N185	-2.74974	-2.64907	-2.55916	-2.49875	-2.44594
IDA_N186	-2.75482	-2.62857	-2.503	-2.48492	-2.43935
IDA_N187	-2.92102	-2.81729	-2.69102	-2.65694	-2.59634
IDA_N188	-2.79064	-2.64407	-2.53938	-2.51061	-2.46562
IDA_N189	-2.86644	-2.72852	-2.64486	-2.59644	-2.55025
IDA_N190	-2.76535	-2.65274	-2.56992	-2.53456	-2.48836
IDA_N191	-2.82767	-2.69113	-2.56424	-2.52854	-2.50268
IDA_N192	-2.74421	-2.58996	-2.51949	-2.47678	-2.44569
IDA_N193	-2.71996	-2.58907	-2.47888	-2.45171	-2.41948
IDA_N194	-2.79703	-2.67058	-2.5859	-2.54315	-2.50827
IDA_N195	-2.78615	-2.64356	-2.53724	-2.49518	-2.45193
IDA_N196	-2.78074	-2.65315	-2.56436	-2.51639	-2.4879
IDA_N197	-2.77092	-2.62677	-2.54959	-2.52238	-2.47503
IDA_N198	-2.80319	-2.67665	-2.5749	-2.5417	-2.51799
IDA_N199	-2.85907	-2.72493	-2.63389	-2.59389	-2.57147
min LLPO	-4.39243	-4.25642	-4.24962	-4.22606	-4.18224

Test set 4	N = 2	N = 3	N = 4	N = 5	N = 6
Files in the comparison set (other non-family viruses):					
IDA_V0	-110.616	-110.613	-110.5	-110.524	-110.546
IDA_V1	-118.424	-118.464	-118.342	-118.365	-118.39
IDA_V2	-120.031	-120.06	-119.949	-119.984	-120.024
IDA_V3	-117.966	-117.968	-117.851	-117.88	-117.901
IDA_V4	-119.481	-119.514	-119.406	-119.432	-119.459
IDA_V5	-120.096	-120.088	-119.978	-120.024	-120.039
IDA_V6	-118.439	-118.484	-118.364	-118.398	-118.42
IDA_V7	-120.106	-120.112	-119.998	-120.029	-120.042
IDA_V8	-120.569	-120.604	-120.489	-120.519	-120.557
IDA_V9	-120.018	-120.051	-119.939	-119.976	-119.99
IDA_V10	-152.121	-152.071	-152.033	-152.041	-152.058
IDA_V11	-101.512	-101.459	-101.352	-101.43	-101.416
IDA_V12	-125.283	-125.202	-125.127	-125.16	-125.158
IDA_V13	-116.656	-116.635	-116.55	-116.628	-116.624
IDA_V14	-101.061	-101.011	-100.905	-100.978	-100.969
IDA_V15	-11.0917	-13.0485	-12.8542	-12.935	-10.9524
IDA_V16	-2.99791	-5.50839	-5.24915	-5.34029	-2.76881
IDA_V17	-10.911	-12.3416	-11.9606	-13.3352	-10.807
IDA_V18	-6.08117	-7.75545	-7.3062	-8.89463	-5.92958
IDA_V19	-2.95419	-5.8593	-5.60426	-5.72121	-2.78321
IDA_V20	-10.1117	-11.3595	-9.83898	-12.1735	-9.94703
IDA_V21	-10.8451	-12.7228	-12.5242	-12.6091	-10.683
IDA_V22	-3.06267	-4.98016	-2.72896	-4.70665	-2.74673
IDA_V23	-3.03823	-5.05128	-2.70359	-4.76206	-2.71525
IDA_V24	-3.04684	-5.02015	-2.7385	-4.75772	-2.75785
max LLPO	-2.95419	-4.98016	-2.70359	-4.70665	-2.71525
Files in the comparison set (random files):					
IDA_R0	-19.1717	-22.3703	-25.4642	-24.4636	-23.4035
IDA_R1	-13.0646	-20.7857	-25.8426	-20.1267	-19.993
IDA_R2	-14.9371	-15.7655	-17.3223	-16.6044	-16.5445
IDA_R3	-20.9474	-21.5938	-27.5827	-26.0192	-21.5414
IDA_R4	-22.7749	-22.8497	-26.9899	-25.9005	-25.2307
IDA_R5	-15.1295	-15.1084	-18.405	-17.9208	-17.9062
IDA_R6	-13.7337	-13.6882	-17.1276	-16.7109	-16.7004
IDA_R7	-14.1933	-14.159	-17.2086	-16.7692	-16.7603
IDA_R8	-15.8097	-15.7774	-19.6769	-19.1926	-19.1698
IDA_R9	-32.1609	-33.0164	-36.13	-33.8791	-33.7931
IDA_R10	-10.1109	-10.1691	-14.3727	-11.3021	-10.1571
IDA_R11	-23.8316	-28.0983	-30.8604	-28.8274	-28.0574
IDA_R12	-8.63557	-8.69344	-17.9714	-13.0733	-12.0628
IDA_R13	-33.6607	-33.6887	-38.6735	-37.9194	-37.8844
IDA_R14	-120.335	-121.546	-125.494	-124.445	-123.859
IDA_R15	-12.1246	-27.6094	-26.5074	-23.5743	-23.5465
IDA_R16	-8.06467	-8.08928	-14.4292	-12.1662	-11.2774
IDA_R17	-13.5816	-13.5934	-15.8965	-13.6325	-13.5501
IDA_R18	-13.0409	-15.7832	-16.3722	-15.0876	-15.0517
IDA_R19	-34.6436	-35.7859	-36.6715	-35.7382	-36.2041
IDA_R20	-33.0675	-33.0497	-36.1826	-34.6595	-37.8778
IDA_R21	-14.1728	-20.2576	-20.9464	-19.5258	-19.4826
IDA_R22	-10.0212	-10.2051	-21.6119	-16.7399	-14.5165
IDA_R23	-10.7864	-10.9172	-17.7176	-16.0938	-12.5916
IDA_R24	-26.7171	-26.8872	-42.853	-40.4542	-37.0516
IDA_R25	-7.47815	-7.45901	-9.58624	-8.55061	-7.46619
IDA_R26	-19.5274	-22.4482	-30.0591	-25.5589	-26.311
IDA_R27	-21.7713	-25.2027	-28.5317	-26.0853	-26.0178
IDA_R28	-14.3515	-14.3586	-21.1881	-18.9919	-17.7641
IDA_R29	-22.0307	-22.0946	-28.2797	-25.2217	-25.0751
IDA_R30	-18.1315	-18.1467	-24.0743	-21.1998	-20.0668
IDA_R31	-22.546	-22.6497	-24.7071	-23.6938	-22.5747
IDA_R32	-31.3673	-43.6283	-47.0231	-43.6668	-43.5965
IDA_R33	-134.343	-134.298	-135.629	-135.014	-134.971
IDA_R34	-37.835	-38.4167	-42.0884	-41.537	-41.5127
IDA_R35	-43.5953	-44.346	-57.5913	-51.4633	-44.3291
IDA_R36	-21.4591	-21.5946	-29.0391	-27.4525	-26.6484
IDA_R37	-17.9073	-18.8366	-23.1441	-22.3652	-23.1984
IDA_R38	-136.477	-137.181	-140.4	-139.155	-138.517
IDA_R39	-38.714	-38.9298	-51.1972	-45.6897	-41.9901
max LLPO	-7.47815	-7.45901	-9.58624	-8.55061	-7.46619