

2013

# VMIFF - Visualization metrics for the identification of file fragments

Ellen J. Hartstack  
*Iowa State University*

Follow this and additional works at: <https://lib.dr.iastate.edu/etd>

 Part of the [Databases and Information Systems Commons](#)

## Recommended Citation

Hartstack, Ellen J., "VMIFF - Visualization metrics for the identification of file fragments" (2013). *Graduate Theses and Dissertations*. 13131.  
<https://lib.dr.iastate.edu/etd/13131>

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact [digirep@iastate.edu](mailto:digirep@iastate.edu).

**VMIFF – Visualization metrics for the identification of file fragments**

by

Ellen Jordan Hartstack

A thesis submitted to the graduate faculty  
in partial fulfillment of the requirements for the degree of

**MASTER OF SCIENCE**

Major: Information Assurance

Program of Study Committee:

Yong Guan, Major Professor

Doug Jacobson

Barb Licklider

Iowa State University

Ames, Iowa

2013

Copyright © Ellen Jordan Hartstack, 2013. All rights reserved.

## DEDICATION

I would like to dedicate this thesis to my family who encouraged me in all aspects of my life, including my education. They've always encouraged me to follow my passions and to work on becoming the best version of me I can.

Thank you.

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b> .....	v
<b>ABSTRACT</b> .....	vi
<b>CHAPTER I. INTRODUCTION &amp; MOTIVATION</b> .....	1
1.1 Introduction.....	1
1.2 Motivation.....	2
1.3 Thesis Overview.....	3
<b>CHAPTER 2. BACKGROUND</b> .....	5
2.1 File System.....	5
2.2 Disk Fragmentation.....	7
2.3 File Fragmentation.....	8
2.4 FAT Corruption and File Deletion.....	10
<b>CHAPTER 3. OUR OBJECTIVE</b> .....	11
3.1 Problem Definition.....	11
3.2 Evaluation Metrics.....	12
<b>CHAPTER 4. LITERATURE SURVEY</b> .....	14
4.1 Introduction.....	14
4.2 File Structure Based Carvers.....	15
4.3 Bifragment Gap Carvers.....	16
4.4 Statistical and Machine Learning Carvers.....	17
4.5 Visualization Carvers.....	17
4.5.1 Byte Plot View (1 byte).....	18
4.5.2 Digraph View (2 bytes).....	20
4.5.3 3D Graph View (3 bytes).....	22
<b>CHAPTER 5. VMIFF DESIGN</b> .....	24
5.1 Introduction.....	24
5.2 Digraphs & 3D Graph Programs.....	24
5.3 Collection of Fragments.....	25
5.4 Fragmentation Program.....	28

5.5 Graphical Binning Program.....	30
5.6 Feature Selection Framework .....	33
5.7 Weka Software .....	34
5.8 Machine Learning Algorithm & Best Features.....	36
<b>CHAPTER 6. EVALUATION AND CASE STUDY .....</b>	<b>37</b>
6.1 Selection of Best Features & Granularity .....	37
6.1.1 Avi Vs. All .....	39
6.1.2 Doc Vs. All .....	40
6.1.3 Exe Vs. All.....	41
6.1.4 Gif Vs. All.....	42
6.1.5 Jpg Vs. All .....	43
6.1.6 Mov Vs. All.....	44
6.1.7 Pdf Vs. All.....	45
6.1.8 Ppt Vs. All.....	46
6.1.9 Txt Vs. All .....	47
6.1.10 Wav Vs. All.....	48
6.1.11 Wma Vs. All.....	49
6.1.12 Wmv Vs. All.....	50
6.1.13 Zip Vs. All .....	51
6.1.14 General Trends .....	52
6.2 Case Study – 13,000 File Fragments.....	52
<b>CHAPTER 7. DISCUSSIONS.....</b>	<b>55</b>
<b>CHAPTER 8. SUMMARY AND FUTURE WORKS.....</b>	<b>56</b>
8.1 Summary.....	56
8.2 Future Works.....	56
<b>APPENDIX A. ARFF FILE FORMAT.....</b>	<b>58</b>
<b>APPENDIX B. 2D AND 3D VISUALIZATIONS OF FILES .....</b>	<b>59</b>
<b>BIBLIOGRAPHY .....</b>	<b>62</b>

## ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who have helped me in all aspects of education and life. First and foremost, I would like to acknowledge my major professor Dr. Yong Guan for his guidance and support in my general classwork and this research. He has provided me with both encouragement towards completing my goal and challenged me to produce my best work.

I thank my committee members, Dr. Barb Licklider and Dr. Doug Jacobson, for their guidance not only on this project, but also throughout the Cybercorps program.

I would like to acknowledge my undergraduate advisor Jason Shepherd from Buena Vista University for mentoring me and challenging me to pursue my education into graduate school. He never let me get away with not knowing or understanding something and always encouraged me to challenge myself.

Benjamin Holland was instrumental in guiding me through the thesis process and encouraging me when the times got tough. He stayed up many a late nights listening to my ideas; providing both constructive feedback and a shove in the right direction when needed. Curtis Putney consistently encouraged me, challenged me and provided constructive feedback to further this work. Both Ben's and Curt's companionship and humor during our late office nights will always be much appreciated.

Finally I would like to thank all my friends, advisors and mentors throughout my life who have provided me with the motivation to keep pursuing my goals.

## ABSTRACT

Visualization of complex data, such as a file system or file, allows a forensic analyst or reverse engineer to rapidly locate areas of interest amidst a large quantity of data. While visualization provides a promising form of analysis, is the subject of much skepticism, as human interaction is required in order for this method to be successful. As a result of this, visualization methods face two major obstacles: tediousness and time.

As our contribution, we propose a unique method of graphing visual information into a measurable format suitable for use with machine learning algorithms. This method will still utilize the visual layout of the data but streamline this form into one that can be rapidly processed by a machine.

In this work we examine existing methods of file fragment analysis, determine how to apply visualization to this analysis, and transform this visual data into a measurable format for machine learning algorithms using our tool called VMIFF (Visualization Metrics for the Identification of File Fragments). In its breadth, this work aims to demonstrate that such transformations will still yield meaningful results.

## CHAPTER I. INTRODUCTION & MOTIVATION

### 1.1 Introduction

In 2009, it was estimated that 250 Exabytes ( $2 \times 10^{21}$  bits) of data would be generated and copied worldwide [29]. In 2010, it was estimated that this figure quadrupled in size to almost 1000 Exabytes a year [29]. With more of our information being stored digitally, criminal investigators have had to refine their skills to adapt to new forms of evidence [17]. Not only this, but they must also hone their skills to be able to sift through all of this information and identify which files will prove valuable to the case. This sorting procedure can take a great deal of time by itself, but if the hard disk or file system has become corrupted this process can become even more time consuming if not completely impossible [22].

Corrupted file systems or hard disks transform the previous useful data stored in these systems into meaningless binary. It is now up to the investigator to carve the relevant information from the file system and reformat it back into its original structure. This is by no means an easy task.

There are many different approaches to recovering files from a disk. Some methods focus on how to recover deleted fragments on a file system. Others take more of a full file system approach and focus on identification of a whole file, file fragment classification, file carving, and/or reassembly of fragments.

This paper will focus on file fragment classification. Specifically we will look at how the visualization of binary file fragments can be useful in classifying them into a



specific file type (ex. doc, jpg). We will be utilizing 2D graphs to visualize these file fragments before transforming these visual structures into a measurable or metrics based form. This constructed data will then be used in conjunction with the Weka data mining program to build a model for each fragment type.

## 1.2 Motivation

A longitudinal study of households from 1984 to 2009 showed an increase in the number of households with a computer at home from 8.2% in 1984 to 61.8% in 2003 [24]. In 2010 the United States Census Bureau found that of a sample size of 119,545 households, 91,724 (76.7%) own at least one computer [25]. Not only are computers becoming more commonplace but the storage capacity of these devices is also increasing. For the average computer, it is becoming more common to find hard drives with 2 terabyte drives or more. These drives are becoming readily available to the public as the price of these components continues to decline (25).

More drive space means more data and more data means more time spent performing analysis on this data. Knowing this, investigators have had to start getting creative in how they approach the analysis of files. Visualization of complex data is an often overlooked and underutilized approach to this issue [6]. It is only recently that recognition of what visual formats can bring to the analysis has been acknowledged.

BlackHat presenters Gregory Conti and Sergey Bratus believe that “the techniques of breaking down large binary objects into simpler parts will be useful for

fuzzing, file carving, reverse engineering, malware analysis, file type identification and other forensics and security analysis tasks [4].”

The downside to existing visualization methods is that these methods require some sort of human interaction in order to provide insight about the file being analyzed [8]. This can lead to an increase in overhead, or the amount of time it takes to correctly classify or identify the key features of a file or file system [22]. While this is still an improvement over manually going through a program or file line by line, Conti and Bratus acknowledge that for this method to be truly effective, regardless of its purpose, it must become automated by the use of classification, clustering or data mining [3].

In order to address this issue we propose a method to capturing the essences of this visualized data into a measureable form which can then be processed by a machine learning algorithm using our VMIFF (Visualization Metrics for the Identification of File Fragments). We hope that by utilizing this method we will be able to bridge the gap between the visual world and the digital. Our method will help to address the amount of time associated with requiring manual human interaction to identify each file fragment by automating this process. We hope that our research will help to further improve the methods of file fragment identification, particularly in regards to visualization.

### **1.3 Thesis Overview**

The rest of the paper is as follows. Chapter 2 - Background will provide the necessary information on how file systems function and how corruption of these file systems can impact the recovery of files. Chapter 3 – Literature Survey will discuss

traditional methods for approaching this problem, like file structure based carvers. Additionally this section will discuss current research being done with machine learning algorithms and in the visualization of complex data. The visualization section will address existing methods for viewing complex data. Chapter 5 – VMIFF Design will discuss the tool we developed to transform a visual representation of the bytes of a file into a format suitable for machine learning algorithms. Chapter 6 – Evaluation and Case Study will apply our VMIFF tool to a simulated file system composed of 13 different file types and evaluate its results. Chapter 7 – Discussion will address potential criticisms of our work. Chapter 8 – Summary and Future Works will provide an overview of this work and a discussion of future works.

## CHAPTER 2. BACKGROUND

### 2.1 File System

In any operating system, the hard disk is divided up into small blocks often referred to as clusters [16]. These clusters are of a certain set size and the files are saved over top of these clusters. For instance if each cluster size is two kilobytes and the file to be saved is eight kilobytes in size, then the file will end up occupying four clusters.

A brand new hard drive begins essentially as a blank slate of empty clusters. There are no files or data written to the drive. As a user or process writes information to the hard disk these clusters become filled with whatever information the file contains.

Figure 2.1 details what a file system might look like. In Figure 2.1 the files “A”, “B”, “C”, and “D” are all neatly organized in the first five clusters of the file system. Notice how file “D” spans two clusters instead of just one.

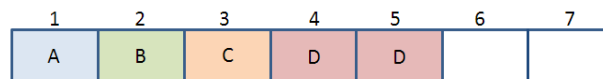


Figure 2.1 – A hard disk with two empty clusters.

When the files are actually saved onto a cluster, the file system needs to know where these files will be located. This will allow the file system to quickly access these files for the user at a later date. For the purpose providing a general overview on how file systems work, we will focus on how the Windows operating systems manages the file system using a FAT architecture. (Please note there are other types of file systems

and that our tool, VMIFF, works with the bytes of file or fragment so the original file system is not considered.)

For the Windows FAT file system, it uses what is called a file allocation table (FAT) to list the clusters occupied by a file [16]. There are generally five ways each cluster is labeled: free (empty), reserved, bad (corrupted), last cluster or next cluster [16].

File Allocation Table (FAT)		
Cluster	File	Type & Pointer
1	A	Reserved
2	B	Reserved
3	C	Reserved
4	D	Reserved -> 5
5	D	Reserved -> End
6		Free
7		Free

Figure 2.2 – The FAT table for the file system in Figure 2.1

According to the file system in Figure 2.1, the FAT would look like Figure 2.2. From Figure 2.2, cluster 3 would be marked as reserved for file “C” whereas cluster 6 would be marked as free as no file has been saved here yet. If we examine file “D” from Figure 2.1 we can see how the FAT directs the file system on how to find larger, multi-cluster files. In the FAT in Figure 2.2, cluster 4 would be marked as reserved and contain a pointer to direct the file system to cluster 5 [16]. This tells the file system where the rest of the file is located. The file system would jump to cluster 5 and continue reading

the file. At the end of cluster 5 there would be a marker that would tell the file system it had reached the end of the file “D” [16].

There are four issues which impact file systems that increases the difficulty investigators face in analyzing them. These four issues are disk fragmentation, file fragmentation, FAT corruption and file deletion. Disk and file fragmentation prohibit investigators from simply reassembling unknown or corrupted files by reading a file system from its first cluster to its last cluster, sequentially. FAT corruption can hinder the recovery of files from a file system because without a FAT to guide the file system on what information is stored where, the information on a disk is nothing more than ones and zeros. File deletion effects investigators in a similar way as FAT corruption does; the data persists on the disk but is no longer retrievable by the file system.

This is similar to removing the table of contents from a book. While all the pages and information are still inside the book, without the table of contents it is difficult to be able to locate any of the chapters. The only way to find a chapter would be to manually flip through each of the pages until you locate it.

## **2.2 Disk Fragmentation**

The computer system has the difficult job of attempting to organize files in the most efficient manner possible. The goal of the computer system revolves around how to fully utilize the disk space the system has available[15]. This goal of efficiency is impeded by the fact that the user constantly modifies, deletes and creates new files. This creates a disk that is essentially riddled with holes of empty clusters where previous files

used to be [16].

Figure 2.3 shows a disk suffering from disk fragmentation. The operating system has had to move, delete and/or modify files many times resulting in this fragmentation. While file “A” has stayed in its original location, all of the other files have been moved around on the hard disk. There is no longer a set of empty clusters located at the end of the file system. Instead of these empty clusters being located at the end of the hard disk, they are now spread about the drive.

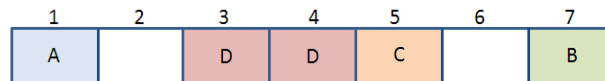


Figure 2.3 – A fragmented disk.

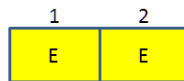
Once a disk has begun to show signs of disk fragmentation, the amount of disk fragmentation present on a hard disk will only increase. This is due to the fact that the file system can no longer sequentially add, delete or modify files onto the empty clusters.

### 2.3 File Fragmentation

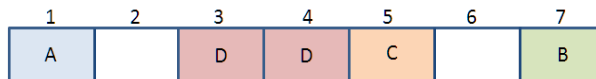
Now that we understand the location of files are stored in the FAT and how clusters work to store a file’s content, we will now look at how saving a larger file to an already fragmented disk will end up creating file fragmentation.

For example, Figure 2.4 (a) is a two cluster long file called “E” which we want to add to the fragmented disk in Figure 2.4 (b). Note that this fragmented system, does not

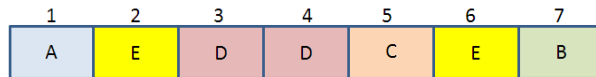
have two empty clusters next to each other. Therefore the file system must split up file “E” into two separate one cluster sized pieces. The file system then stores the first part of file “E” into the empty cluster 2 and the second part of “E” into the empty cluster 6. This results in the file fragmentation of “E” in Figure 2.4 (c).



(a) File “E” is 2 clusters in size.



(b) Fragmented disk where “E” is to be added.



(c) The resulting file fragmentation of “E”

Figure 2.4 – A fragmented disk which upon adding file “E” demonstrates file fragmentation.

The example from Figure 2.4 only has a file system size of seven clusters. As you can imagine, real file systems are composed of billions of clusters filled with millions of files. Therefore from this small example you can begin to see the implications of how file systems can easily become susceptible to disk and file fragmentation.



## 2.4 FAT Corruption and File Deletion

If the FAT becomes corrupted or a file is deleted, the original file data still persists on the hard disk but it can no longer be easily retrieved by the file system.

When a file is deleted the file system simply marks the clusters currently occupied by this file as being empty and available for new files to be written to in the FAT [16]. Even though these clusters are still technically filled with the information of these now deleted files, the file system has removed the entries from the FAT so the file system no longer has the pointers to where this file used to be located at. In either of these two situations the file system is now no longer able to locate these file, even though the data of the file is still located on the hard drive, because all mention of which clusters contain the relevant file/s have been removed.

## CHAPTER 3. OUR OBJECTIVE

### 3.1 Problem Definition

Recovering a file that has been fragmented presents a problem to investigators due to the fact that files are identified by most file carvers by using the file's signature. When viewing a fragmented file only the first piece of that file will contain the file's header and only the last piece of a file will contain the file's footer. File systems are only able to understand the order of the middle pieces of a file and to which file these fragments belong due to the pointers located in the FAT. These pointers allow the file system to reassemble the file when a user or process requests it. However when a file is deleted or the FAT becomes corrupted, all record of where the individual middle pieces of that file are located have been removed.

For our work, we assume the worst case scenario where there is no structure or file signatures to base our file recovery efforts upon. We intentionally ignore both the header and footer fragments in order to focus our file type identification to satisfy this situation. Therefore all we have available for analysis are the raw bytes of the file fragments with the first and last cluster removed.

Given these bytes, we assume the smallest possible byte size of 512 bytes. The sector size of 512 bytes was selected as cluster sizes are traditionally in multiples of 512 bytes [19]. Thus by setting our fragment size at 512 bytes we have addressed the smallest available fragment size on most systems. We also assume that these files have

been fragmented and are no longer in the sequential order necessary for traditional recovery.

This situation will help to ensure that we are focusing our identification efforts on only mid-fragments, the most difficult to detect as they contain no information linking them back to either the type or the original file.

Traditionally there are three different types of file carving problems. The first problem is: given a set of fragments identify the file type of these fragments. The second problem is: given one file whose fragments are out of order, determine the correct order of the fragments to reassemble the file. The last file carving problem is: given a set of fragments of multiple types and non-sequential order, determine both the type of file and the correct order for reassembly.

For our research we will be focusing on the first problem: identification of file fragment types. The other two problems will be discussed in Chapter 8. We selected the first file carving problem as a means to illustrate a positive correlation between the visualization of data and our VMIFF tool's ability to correctly capture this.

### **3.2 Evaluation Metrics**

Two types of metrics were used to evaluate our results. The first metric is associated with how we measure and transform our visual data into a measureable form via VMIFF which is suited for being processed by a machine. This was measured using two variables: features and granularity. Features are the algorithms we used to capture how the data looks visually into a measureable format. For example a feature might be

the number of points in a specific area of a graph. Granularity refers to the how closely we analyzed the data. A low granularity means we are looking at a larger portion (big picture) of the graph while a high granularity would correlate to a smaller section of the entire graph.

The second type of metric relates to how well our binning program worked. We measure the successfulness of VMIFF with three variables: true positive (TP), false positive (FP) and overall run time. These three components will be utilized in both Chapter 5 – VMIFF Design and Chapter 6: Evaluation and Test Cases.

## CHAPTER 4. LITERATURE SURVEY

### 4.1 Introduction

The process of utilizing file carving in order to recover deleted files or those files from a corrupted FAT file system can become much more complicated if the deleted file has been subject to file fragmentation or if the drive itself has become fragmented. Popular file types like AVI, DOC, and JPG, all stand to have significantly higher occurrences of fragmentation than other less used file types like BMP, HLP, INF, and INI [13]. This is both due to the frequency in which they tend to be modified (doc) and that the larger file is the more likely it is to be fragmented [13]. In regards to how this might affect forensic investigators these popular highly fragmented file types are some of the most commonly examined file types when investigators inspect a hard disk. [13].

The process of recovering files is currently hindered because the existing methods of how tools perform file carving have two main obstacles they have yet to overcome. The first obstacle is that a majority of today's popular carving tools carve out files utilizing sequential clusters [13]. This fails to account for the issue of fragmentation where empty clusters or other files may exist between the fragments of the desired file. This paper will attempt to address this issue.

The second major obstacle is that tools rarely check the files they have managed to recover for the validity of the file itself [13]. Therefore these types of tools end up recovering a lot of files that have been corrupted. The problem stems from the fact that the tool fails to validate if the file has been successfully reassembled in the correct order

with all of the correct fragments. Additionally a lot of file carving tools will present the user back with files that are already listed in the FAT, which are uncorrupted and easily accessible via the traditional file system. Thus this only presents the forensic investigator with more files and more data to sift through without dramatically increasing the number of useful files. This issue will be discussed further in Chapter 8.

There are four general types of carvers that are currently used by the forensics community to recover fragmented files. These types are: file structure carvers, bifragment gap carvers, statistical and machine learning carvers, and visualization carvers. The focus of this paper will be to utilize visualization as well as statistical and machine learning carving methods to assist in the identification of file fragments.

## 4.2 File Structure Based Carvers

Carvers that work based on the overall structure of the files are the most common of the four types. This method utilizes known header and footer information to carve out the file [19]. For example a jpeg header will start with the hex sequence of FFD8 and will end with a footer hex sequence of FFD9 [19]. Once these two location markers have been identified all of the fragments between these two markers would be carved out and then reassembled back into the original file type [19].

For some file types additional information like file size might also be included in the file's header information [19]. This information could then be used in the validation of a file after it has been carved from the file system [19]. However if the fragments of the file were out of order, the validation check would pass successfully but the resulting

carved file would still be corrupted.

These carvers can also be adapted to fit file types like ZIP which only have a file signature header but no footer [13]. This method works by identifying a header signature and then gradually expanding out one block at a time [13]. After each expansion the resulting file is then validated to see if it matches the expected overall structure of that file type [13]. If not, the file is expanded again and revalidated until a successful validation occurs [13]. The main issue that stems from this signature based file carving method is that it ignores the fact that the fragments of this file might not be in sequential order [19].

### **4.3 Bifragment Gap Carvers**

In an attempt to address the issues with the previous type of carver, Simon Garfinkel developed the bifragment file carver which assumes that the file exists in two fragments with a gap of unknown size between them [13]. This method is similar to a guess and test method. It identifies the starting point and ending point of a file via the file signatures, aka the header and footer, and then gradually increases the gap between these two sections until a valid file can be carved [13].

This method also relies heavily on the fast object validation technique developed by Simon Garfinkel in order to address the issue of incorrect validation [19]. This method works by checking to see if a carved file adheres to expected overall structure or rules of that particular file type [19]. What this method fails to account for is the fact that file may exist in more than two fragments.

#### 4.4 Statistical and Machine Learning Carvers

These carvers take a different approach to categorizing fragments into their appropriate file type than the first two types of carvers. This method looks at the overall structure of the file fragment versus being heavily dependent on the header and footer signatures [21]. These methods rely on a statistical measurement or features of a file fragment to help identify a fragment's type. For example one feature is known as a byte frequency distribution (BFD). This feature looks at the frequency in which each byte appears in the fragment and displays this information into a vector [21].

After collecting a variety of features about a certain file fragment type, the information can then be combined into a blueprint of what file fragments of this type should look like [21]. Machine learning algorithms work well with this approach. A training set will be created to “train” a machine to learn which features should be used to identify a fragment as a specific type [21]. Once this model has been build, a test set of fragments will be generated to measure the effectiveness of the model at predicting the correct file type of new fragments. Typically one feature alone cannot be used to solely identify the type of the fragment. Thus features are often used in conjunction with other features in order to make a unique profile for each file type [21].

#### 4.5 Visualization Carvers

Existing visualization carvers tend to be geared towards the reverse engineering (RE) profession. These carvers will provide a visualization for an entire file system or file. Typically they also provide additional tools that will serve the RE profession like



hex editors, encryption and decryption methods, checksum calculators, and compression and decompression components [6]. Additionally many of the more sophisticated tools will provide general metrics on the file or file system being analyzed like a byte frequency histogram [6].

There are three general classes of visualization tools: byte plot, digraph and 3D. Byte plot views look at each byte of the file individually, digraph views use two bytes to form points on a graph and 3D visualizers use three bytes.

#### 4.5.1 Byte Plot View (1 byte)

During the 2010 BlackHat conference, Sergey Bratus and Greg Conti gave a presentation regarding the visualization carver they had built. Their carver operates by reading in each byte of the file or file system and coloring them based on each bytes' numerical value [3]. In the Figure 4.1 below the color white would be represented by the byte value of 255 and the color black would be represented by 0, with a scale of grays between these two values [3]. The bytes are ordered from left to right as they are read in, meaning that position (1, 1) is the first byte and position (1, 2) is the second [3].

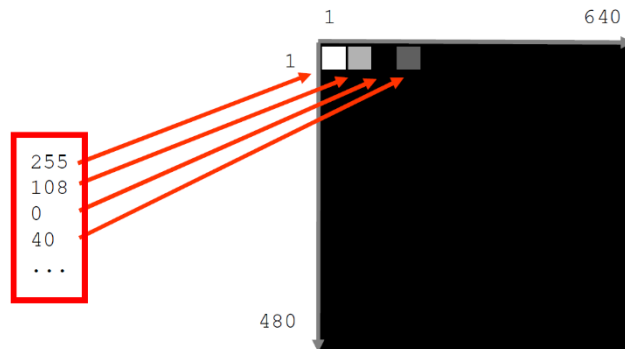


Figure 4.1 – Sergey Bratus and Greg Conti Visualization Carver.

This type of carver will read through the entire file or file system and create a visual structure of what the file or file system might look like. In Figure 4.2, the jar file tools.jar has been completely read in by the carver and mapped into this visual structure [3].

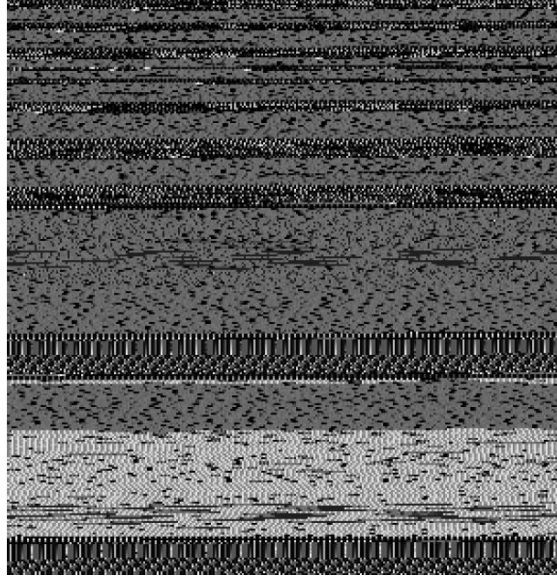


Figure 4.2 – The resulting visualization of a tools.jar file.

In their paper Bratus and Conti discussed how by viewing a file in this manner enables the reverse engineer to be able to visually locate areas or sections of interest to focus on during their investigation [4]. This would speed up the investigative process because the reverse engineer is no longer having to browse through a program line or memorize complex patterns of hex. This is all because they can now visually identify areas of potential significance and focus their investigation there.

Bratus and Conti demonstrated that investigators could be trained to identify the overall structure of the type of data they were looking for based on sample data [3]. For

example if the investigator was looking for code written in C++ he or she would be able to identify what C++ code looks like in comparison with other textual types files (see Figure 4.3) [3].

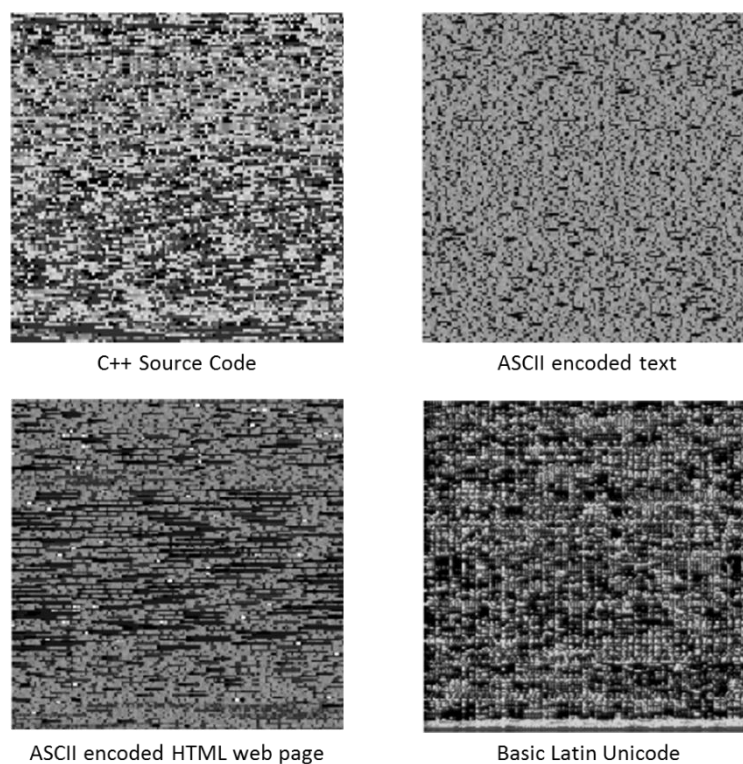


Figure 4.3 – Textual file differ based on their content when visualized

#### 4.5.2 Digraph View (2 bytes)

Another method of visualizing a file discussed by Bratus and Conti was to plot pairs of points to into a digraph view [3]. For example, the word “visual” would be represented in bytes as {118, 105, 115, 117, 97, 108}. These bytes would then be paired

into coordinates to be mapped on to the graph:  $\{(118, 105), (115, 117), (97, 108)\}$ .

These points would then be plotted on an x and y graph (see Figure 4.4).

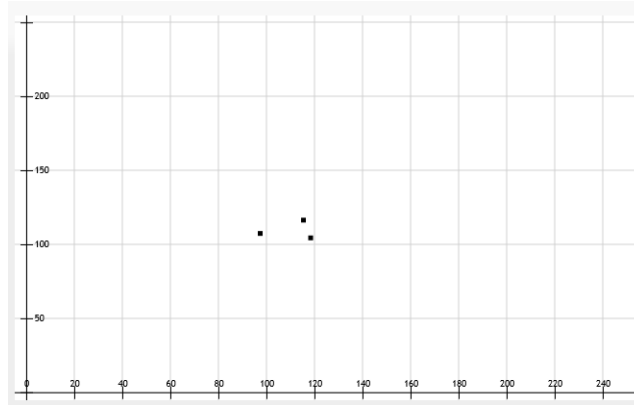


Figure 4.4 – A graph of the byte representation of the word “visual” [26]

Bratus and Corti then visualized an entire text file. Figure 4.5 is a representation ASCII encoded English text.



Figure 4.5 – ASCII English text displayed in digraph view.

### 4.5.3 3D Graph View (3 bytes)

During the 2012 DerbyCon security conference, Christopher Domas of Battle Memorial Institute presented “The Future of RE: Dynamic Binary Visualization” detailing a program called cantor.dust [8]. While cantor.dust is again specifically tailored towards reverse engineers it offers a number of different ways to look at binary data visually. Cantor.dust utilizes both color, bytes and byte patterns in order to help display the file, file system, memory dump, or packet visually [2]. In Figure 4.6, two files have been transformed into a 3D graph.

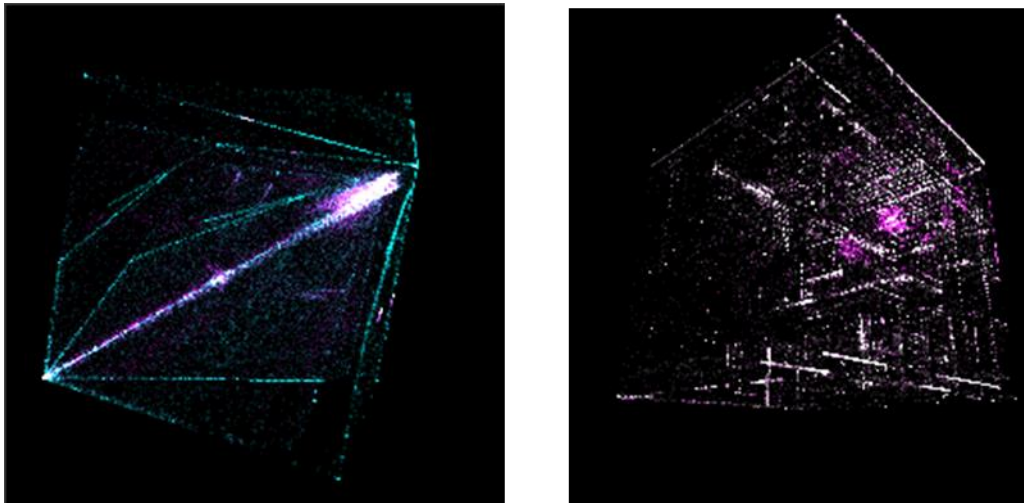


Figure 4.6. Two 3D graphs of two files [2].

During Christopher Domas’ talk, he mentions that this type of software is designed to illustrate a concept versus marketing the visualization software as a solution [8]. Therefore the importance of discovering new ways to visualize data is stressed while

the details on how the software actually processes the data was only briefly mentioned [8]. Therefore this software is not open-source nor available for public purchase [8].

As a result of these facts, little more is known about how the data of a file or file system is exactly processed other than the fact that their tool uses the fractal pattern known as the Hilbert Curve developed by David Hilbert in 1891 (Figure 4.7) to assist in the building of the graph [8].

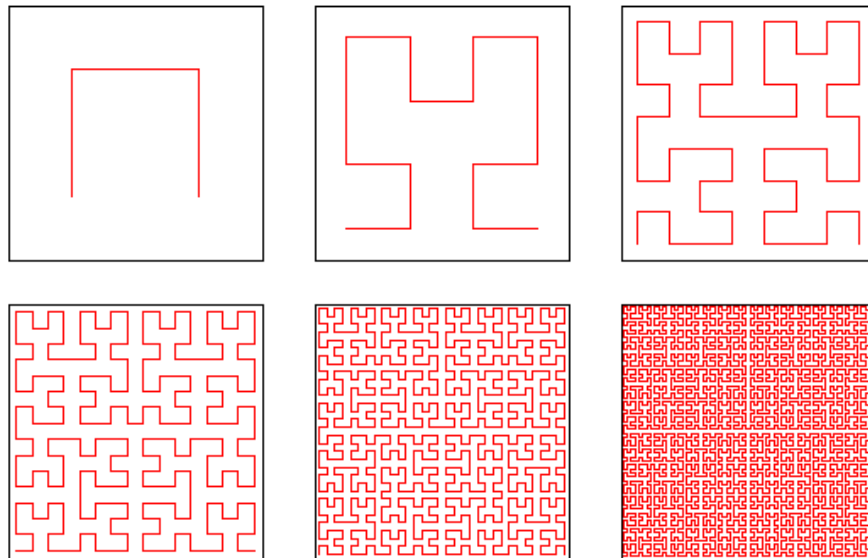


Figure 4.7 – The first six stages of the Hilbert Curve [11].

## CHAPTER 5. VMIFF DESIGN

### 5.1 Introduction

In order to transfer our binary fragments into the visual realm, we first had to develop a digraph and 3D graph program. Our program is called VMIFF (Visualization Metric for the Identification of File Fragments). This program would need to be able to accept a variety of different file formats and be able to display them. Next we needed a variety of file fragments to perform analysis on. The fragments we selected were mid-fragments, meaning that these fragments are located between the file signatures as these types of fragments are considered the most difficult to detect through the use of traditional file structure carvers. Finally we developed a method for transforming the visual graphs into a set of numeric values to represent each file type. Using these numeric values we utilized a machine learning decision tree to automate the file type identification process. Our project will be released as open-source software, VMIFF, on github, in order to allow others to expand on our progress.

### 5.2 Digraphs & 3D Graph Programs

We utilized a pre-existing java chart library known as JFreeChart developed by the Object Refinery Limited Company to help us visualize our binary fragments in a 2D digraph [18]. JFreeChart is free, open source software released under a GNU Lesser General Public License [18]. This library allows for 2D graphs to be generated and displayed by using the existing API calls. Additionally this library also allows for the

customization for each chart via the GUI with zooming, color properties and other display options easily accessible [18]. We selected a traditional scatterplot for our 2D graphs.

For our 3D graphical representation of the binary file fragments, we elected to use the JZY3D open source java library [20]. This library allows for 3D graphs to be represented in a variety of different manners, like scatterplots, bar charts, surface charts etc. [20]. This program was not as in-depth as JFreeChart but still allowed us to represent the binary file fragments on a 3D plane. We also selected a 3D scatterplot view to represent our 3D graphs.

### 5.3 Collection of Fragments

According to the FileExt database, to date (March 2013) they have currently cataloged over 51,537 different file type records [23]. With this many file types available we needed to narrow down our search to include only those whose identification would most benefit the investigators. We selected 13 different file types to be used in our program. These 13 file types were selected both based on their popularity and upon how the larger the file is the more likely it is to become fragmented [13]. More specifically word document files (.doc), audio video interleave files (.avi), and joint photography experts group (.jpg) files were noted to both be of high significance to the forensic investigators and also to have a higher likelihood of being fragmented [13]. See Figure 5.1.



File Fragmentation by File Type					
Extension	# of Files Analyzed	Number with 2 fragments	Number with 3 fragments	Number with 4+ fragments	Percentage fragmented
avi	998	17	6	185	20%
doc	7,673	209	65	1,100	17%
exe	78,646	2,352	827	8,648	15%
gif	357,713	2,990	795	27,581	8%
jpg	108,539	2,999	400	13,973	16%
mov*	--	--	--	--	--
pdf*	--	--	--	--	--
ppt	1,120	20	6	73	8%
txt	64,315	496	125	6,726	11%
wav	24,550	584	143	1,721	9%
wma*	--	--	--	--	--
wmv*	--	--	--	--	--
zip*	--	--	--	--	--

Figure 5.1– Analysis of fragmentation in the wild by Simon L. Garfinkel [13].

\*Note: These files were not included in Garfinkel’s sample file set. However we felt these files to be of significance to investigators and will include them in our test sets.

These file types include: Audio Video Interleave files (.avi), Microsoft Word Document (.doc), Windows Executable file (.exe), Graphical Interchange Form (.gif), Joint Photographic Experts Group (.jpg), Apple QuickTime Movie (.mov), Portable Document Format file (.pdf), PowerPoint Presentation (.ppt), Plain Text file (.txt), WAVE Audio file (.wav), Windows Media Audio file (.wma), Windows Media Video file (.wmv), and Zipped file (.zip) [12].

Once these types were selected, enough files of each type were obtained to create 100,000 fragments of 512 bytes of each type. This would ensure we would have enough

data to both train and test our machine learning algorithm without biasing it to data it had already analyzed.

These files were obtained from two main sources. The first source was the corpus of 1 million files collected specifically for the file fragmentation research [7]. Digital Corpora located these documents by selecting a random word from the UNIX dictionary and combining them with random numbers for documents of specific file types residing on webservers within the .gov domain [7]. These files composed a majority of the files used in the training and test sets.

For those file types which were not included within the above corpus or those file types which did not have enough quantity of in order to fragment into the required 100,000 fragments, a simple Google search was done to locate the missing types. These files were then downloaded and added to the collection of files.

After we had collected enough data, we sorted the files into categories based off of what their file extension was listed as [27].

We then selected files (of similar size) from each of our file types and visually examined it in both a digraph (2D) view and a 3D graph view. The results are listed in Appendix B – 2D and 3D Visualizations of Files.

We then built a fragmentation program to split each file into a 512 byte fragment.

## 5.4 Fragmentation Program

The first and last fragments of most file types can easily be identified and located by traditional carvers, and thus these fragments were not included in our tests. Our process was as follows (pseudocode):

1. Select a file.
2. The first 512 bytes of the file would be skipped. This would effectively skip the first fragment including the file signature's header.
3. Then the file would read 512 bytes into a byte stream buffer and output the resulting fragment.
4. This process would continue until there were less than 512 bytes remaining. This would effectively skip the last fragment which would include the file signature's footer.
5. Then the next file in the folder would be selected for fragmentation.
6. This process would continue until there were 100,000 fragments of each file type.
7. Files of less than 512 bytes were discarded as these would be located in only one cluster and could be carved out by traditional sequential file structure carvers.

After we had successfully fragmented 100,000 of each file type, we selected some random files from the batch to see if the general trends we noticed in visually examining the entire file (Appendix B) still existed in the fragments. Naturally, we assumed the fragments would not exhibit all the trends we noticed when we viewed the original file in its entirety, but we hypothesized that there would be some trends that remained.

For example in Figure 5.2 we can see that although there are fewer bytes of the text file to view visually, the bytes still tend to cluster together in two straight lines with a cluster of points in a square shape in the center.

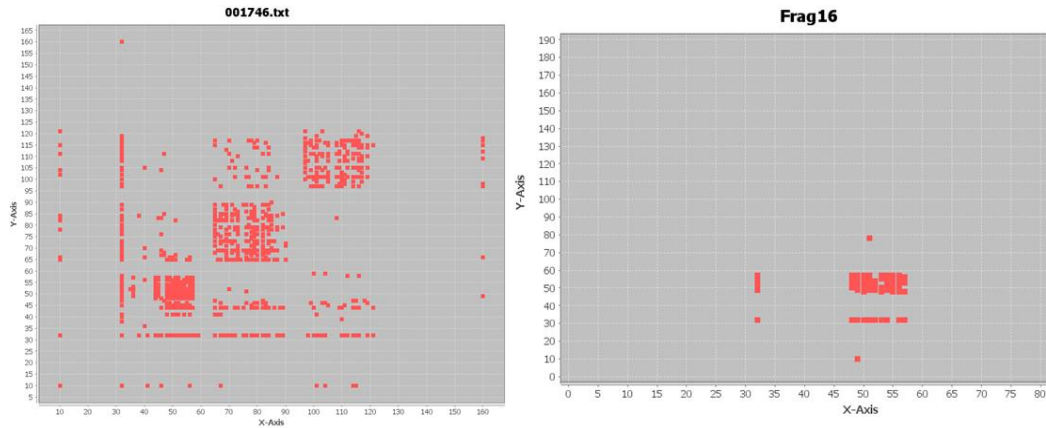


Figure 5.2 – A comparison between the visual representation of an entire .txt file (left) and a randomly selected .txt file fragment of only 512 bytes (right).

Another example was with .wav files. When viewing the whole file visually, the trend seems to be that the points cluster around the center diagonal axis. When looking at a smaller fragment of just 512 bytes, the trend remains although is less predominate (see Figure 5.3).

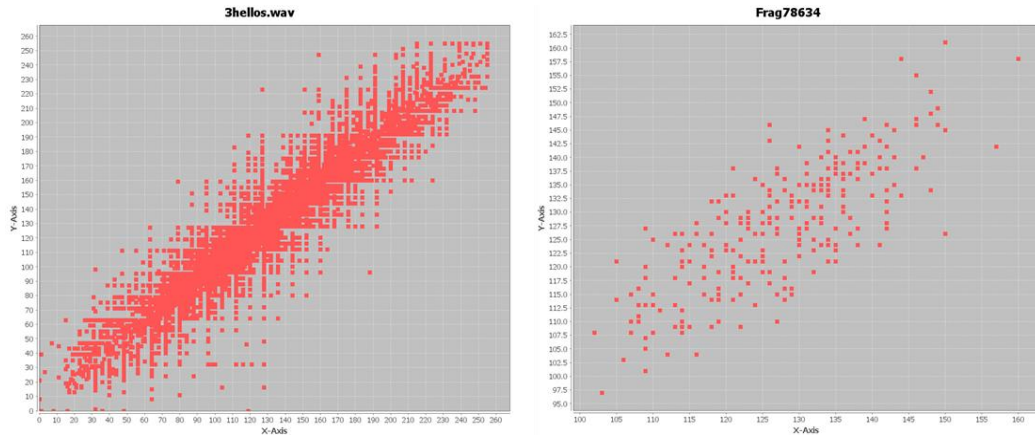


Figure 5.3 – A comparison between the visual representation of an entire .wav file (left) and a randomly selected .wav file fragment of only 512 bytes (right).

While these differences seem to be minor, we hypothesized that if we could properly capture these visual trends we would be able to not only automate the identification of the fragments but also be able to form generalizations about each file type.

### 5.5 Graphical Binning Program

In order to capture these small trends that occur with each of the file types, we needed to develop a method to capture the information contained in the graph that would be meaningful when being processed by a learning machine algorithm. Creating this translation would not only decrease the time it would take to analyze an entire file system of fragments but might be able to identify trends the human eye might overlook.

As with all visualization processes, a human does still need to be involved in the training of the VMIFF tool. We needed a way to cluster the various points together into

small groupings in order to capture the overall “look” or trend of the file. We did this by utilizing a “grain and bucket” approach. Grains or granularity refers to the how many sections or buckets the graph will be divided into. Each bucket will contain information about all the points located within that section.

The granularity (grains) of the graph is calculated as  $4^n$  where  $n$  is a value between 0 and 8. Granularity determines the number of equal sized sections or buckets the graph has been divided into. For example if a graph had a granularity of 0 it would be displayed as one large graph with one bucket to hold all the points like graph on the far left in Figure 5.4. The middle graph in Figure 5.4 has a granularity of 1 and therefore is divided into 4 buckets as depicted by the light blue line. The points of each section are now represented in a set, which starts in the top left quadrant and reads left to right, top to bottom. The center graph’s points would be represented by the set: {3, 1, 0, 2}. The right graph has a granularity of 2. There are now 16 buckets which are represented by the set: {1,0,1,0,1,1,0,0,0,0,1,0,0,0,1,0}.

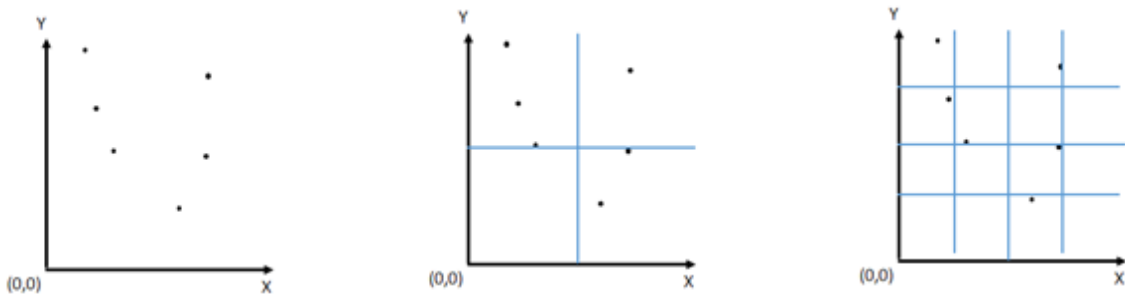


Figure 5.4 – Granularity 0 (left), Granularity 1 (middle), Granularity 2 (right)

Points contain two types of information. The first type is location. Location is represented by the value of the two bytes used to generate that point. The second type is the sequence of that point. Sequence is determined by when the point was added to the graph as it was read from the byte stream. For example the first two bytes are read in and form a point with a sequence of 0, while the last two bytes of the stream will create a point with a sequence of 255.

Graphically data can also be represented in dimensions. Dimensions refers to how many bytes are used to represent a point. If two dimensions are specified then two bytes would be used to form a point. If four dimensions were used, then four bytes would be used to represent a point. For the purposes of demonstrating the positive correlation between visualization and our graphing “grain and bucket” metric, we will be assuming only 2 dimensions will be used although VMIFF will currently accept up to 16. This was done to avoid complicating our results too many variables.

We now have a way to visually represent the graph by using buckets, capture groupings of points by changing the dimensions, and can expand and narrow our focus by modifying the granularity of the graph. Next we need to determine which features or algorithms to run against these graphs in order to capture the visual data into a format suited for our machine learning algorithms. We will need to do this process for each of the 13 file fragment types we selected.

## 5.6 Feature Selection Framework

As with all visualization methods, some human involvement with the data is necessary to take advantage of what makes visualization a useful metric. Humans provide meaning and context to a situation which machines cannot (unless we give them this context). Feature selection is the context. By creating these features we are teaching the machines how to “visualize” as we do. Thus as more file fragment types are added, the feature creating process will need to be modified in order to visually capture these new types (if existing features fail too).

Thus when we designed our feature selection framework, we created a dynamic module approach that makes it easy to program new features to run against our graphical binning program. It is our hope that for future work, we (and others) will be able to expand the number of features available and further develop the ways to capture the essence of a binary file visually.

For the purpose of this paper we selected three features: total points, total sequence, and average sequence. The total points feature calculates the total number of points in each bucket. The total sequence adds the sequence value of all the points in the bucket. The average sequence points returns an average sequence value for all the points in the bucket.

Now that we have developed a way to “measure” how a file fragment looks visually we need to transform this data into a format accepted by a learning machine.



## 5.7 Weka Software

We selected the Weka data mining software to help classify our file fragments. Weka is a java application that contains a collection of machine learning algorithms to assist in data mining tasks [14]. The software was developed by the Machine Learning Project at the Department of Computer Science at the University of Waikato in New Zealand [14].

The first obstacle was to translate our generated metrics into the proper format for Weka. Weka takes in a textual comma delimited Attribute-Relation file format or .arff file. While these files can be typed manually we developed an automated process (Arff Generator) which generates these arff files in the proper format. Arff files use the “@” symbol to indicate the different sections of the file.

For our project we used three sections:

- @Relation- to give the name of the file
- @Attribute - to list each of the features we were using
- @Data - to list each of the files and its corresponding metrics

A sample .arff file has been included in the Appendix A.

When generating these arff files we built a small scripting program which would allow us to automate the arff file generations. Users would type the commands into a text file and then load the file into our arff generator. This program would then read in the commands from the text file and generate the arff file without any need for human interaction.

A command is a series of strings which are separated by “:”. For example:

```
[AVI],[DOC EXE GIF JPG MOV PDF PPT TXT WAV WMA WMV ZIP]:AviVsAll:1000: 5:[TotalPointsFeature]
```

There are five components of each command string. The first details which sets of fragments the user would like to include in the test set. In the above example we are comparing two groups. The first group will consist only of avi fragments and the second will be composed of an equal number of doc, exe, gif, jpg, mov, pdf, ppt, txt, wav, wma, wmv, and zip fragments. This section can contain any number of groups.

The second command is the file name of the outputted arff file. The third command is the number of fragments we want for each of the groups. For the example above there will be 1,000 of each group, so 1,000 avi fragments and 1000 split into 12 groups or 83 of each of the fragments in the second group. This will result in a total of 1996 fragments. The next number tells what granularity we want to view the graph at. This value must be between 0 and 8. The final command tells the arff generator which features to run against each of the fragments. Multiple features can be combined (space delimited), however for the purposes of this paper we will be only looking at the effectiveness of each of the three features separately.

After the arff files were generated, we used the GUI version of Weka and used the Explorer application to run against the data we had collected in our arff files. We used the “classification” component of Weka which allowed our data to be compiled using a known machine learning algorithm. Now that we have loaded our arff file into the Weka software, we needed to decide which algorithm to use to help classify the data.

### **5.8 Machine Learning Algorithm & Best Features**

We elected to use the J48 decision tree as our classifier as this is a well-known and documented classifier [9].

## CHAPTER 6. EVALUATION AND CASE STUDY

After creating up our software, VMIFF, we wanted to evaluate our results on a test case. We selected 1,000 fragments of each of the 13 files types in order to build 13 models to identify each of the 13 file types. These models have a specified granularity and feature which we determined to best fit that particular file type using predictive 10-cross-fold validation. We then built a test set of 13,000 fragments to run against this model. The results will be detailed and examined in the section below.

### 6.1 Selection of Best Features & Granularity

The first step of building our models was to determine which granularity and features to use. We determined that an exhaustive comparison of all combinations of both granularity and features would be the most effective manner to determine the “best” fit model.

When considering which feature and granularity we were going to use to build the model from we considered multiple variables. The first was to examine which combinations of granularity and features resulted in the highest true positive rate. The other variable was to consider the computational time for each grain. While our results were generated on multiple computers, we noticed that computational time as well as file size dramatically increases as the number of grains increase (Figure 6.1)

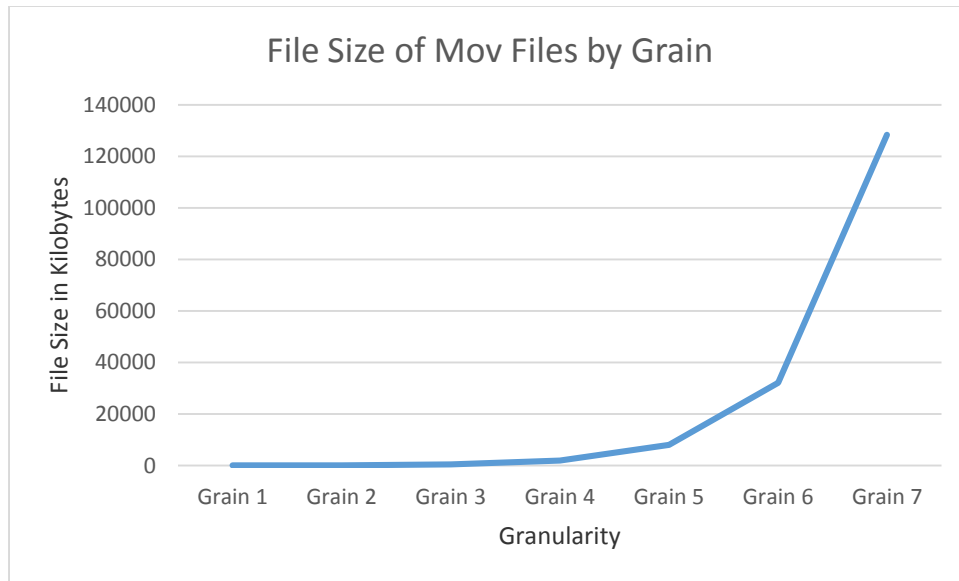


Figure 6.1 – Demonstration of file size growth based on granularity

We would be using our three features: average sequence, total sequence and total points for our test case. For granularity we would be using grains 1 through 7. Grain 0 was eliminated as it would return the same values for all fragments and grain 8 was eliminated as it would return the entire graph and be extremely computationally intensive.

Our data set contained 13 models. Each model had the file type we were going to identify compared to an equal sized collection of all other 12 file types. We used a total of 1996 fragments in the training set: 1000 of these consisted of the file type we were trying to identify and 996 were composed of an even mixture of the 12 file types (83 fragments of each of these types).

The results of each model are detailed below listed alphabetically by file type. We include both a table and graph of the results and a brief summary of our findings.

### 6.1.1 Avi Vs. All

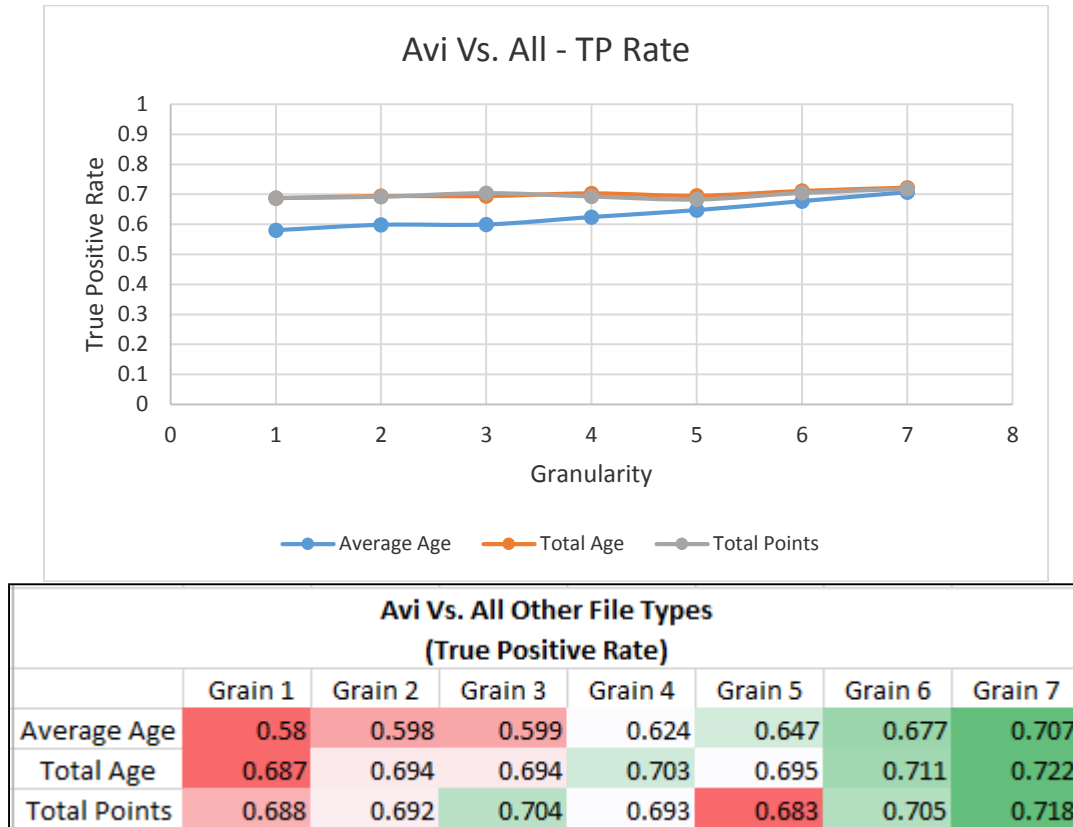


Figure 6.2 – The true positive rate of avi fragments vs. other fragment file types

The general trend of this data is that true positive rates slightly increase as the amount of grains increase. While it was determined experimentally Total Sequence – Grain 7 offered the highest results, we elected to use Total Sequence - Grain 6 for our avi model. We felt that a true positive rate increase of 1.1% was not enough to merit the increase in runtime between grain 6 and grain 7.

## 6.1.2 Doc Vs. All

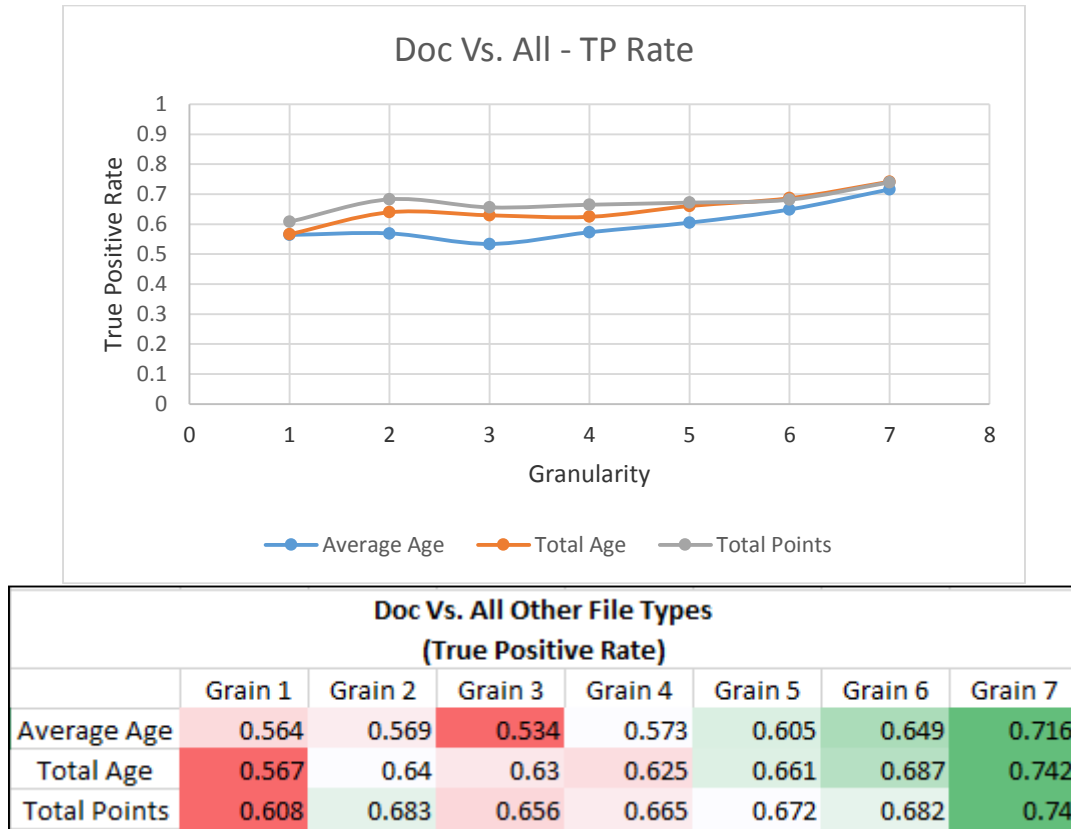


Figure 6.3 – The true positive rate of doc fragments vs. other fragment file types

The general trend of this data is that true positive rate increases as the amount of grains increase with a slight decrease near grain 3. When looking at only 512 bytes of a word documents it can be difficult to determine if the fragment is of a doc file, if this doc file has other objects embedded within it (ex: a jpg image) [5]. Therefore a portion of those fragments misidentified could have originally been a doc file but that particular 512 byte fragment was taken from an embedded jpg object in the word doc [10]. Word documents can greatly fluctuate in their byte representation based on what type of

information they contain [10]. For instance a word doc with only text has an average byte value of just 90 while a word document containing images has a higher average byte value of 125 [10]. These factors can increase the difficulty in detection. We selected Total Points – Grain 7 for our doc model, as the true positive rate between grain 6 and grain 7 was 5.5%.

### 6.1.3 Exe Vs. All

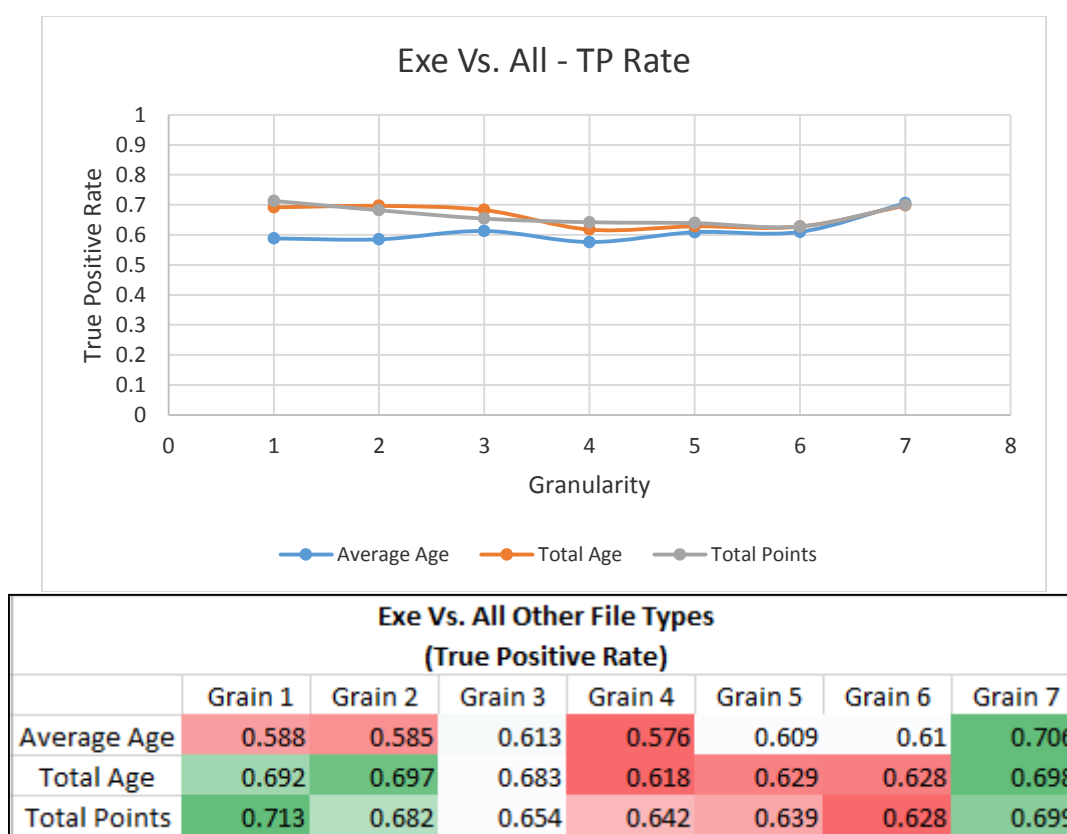


Figure 6.4 – The true positive rate of exe fragments vs. other fragment file types



The general trend of this data is that for both total sequence and total points the true positive rates generally fall into a parabolic curve with the highest values being in grain 1 and grain 7. We selected Total Points – Grain 1 for our exe model as it was both the highest true positive rate and would result in the shortest amount of run time.

#### 6.1.4 Gif Vs. All

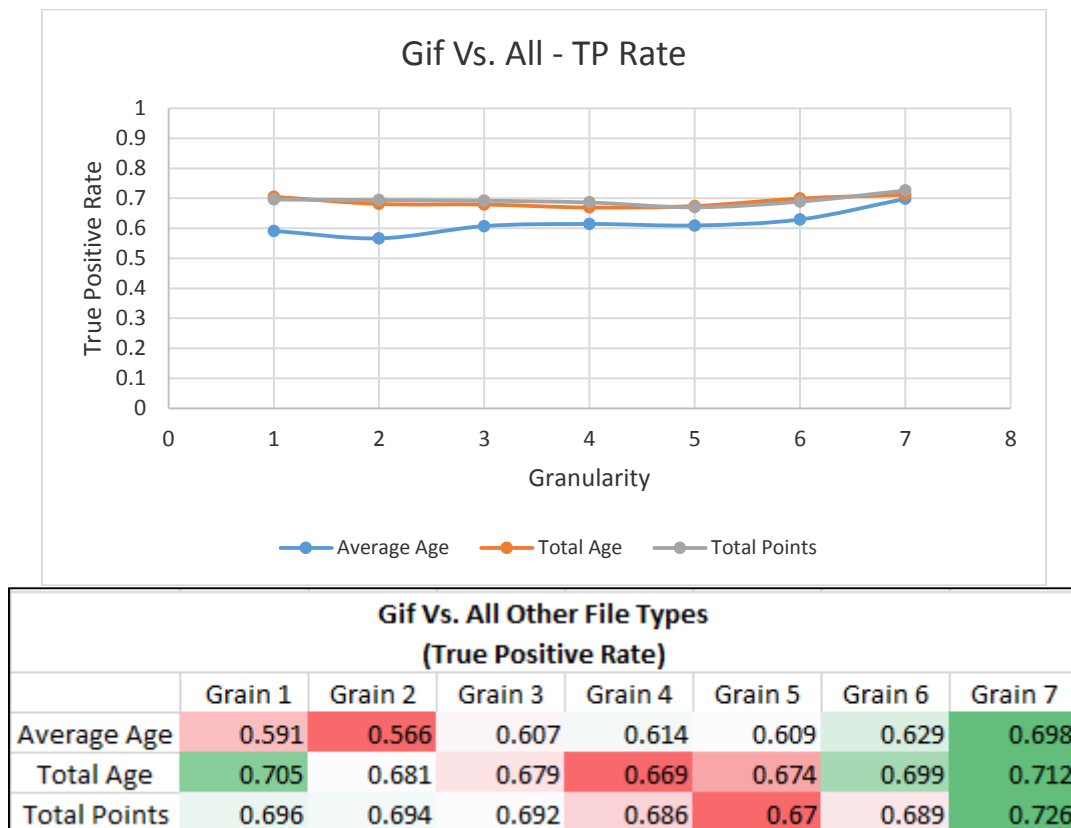


Figure 6.5 – The true positive rate of gif fragments vs. other fragment file types

The general trend of this data is that regardless of the feature used the true positive rate generally plateau. While it was shown that Total Points – Grain 7 was

statistically the highest true positive rate, we selected the Total Sequence – Grain 1 for our gif model as grain 1 has a dramatically shortened run time.

### 6.1.5 Jpg Vs. All

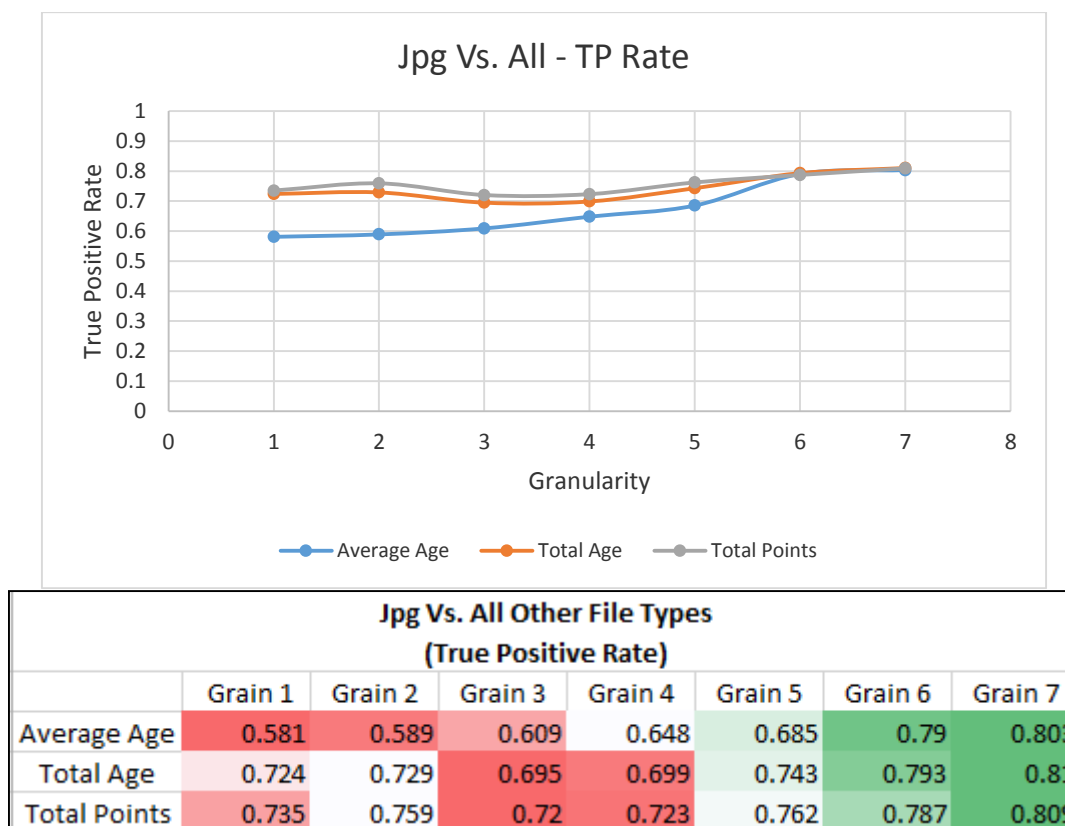


Figure 6.6 – The true positive rate of jpg fragments vs. other fragment file types

The general trend of this data is that true positive rate increases as the amount of grains increase. Therefore as the number of grains increased the better the VMIFF tool was able to capture this trend. Thus for the jpg file type we selected Total Sequence – Grain 7 for our jpg model.

Jpg file types are very organized file structure compared to other file structures. Jpg files are uniformly organized giving them an average byte value between 120 and 142 [10]. This fact helps to aid in their identification as the only part that is not uniform is the very beginning of the jpg file (which is removed by our VMIFF tool when the header was removed) [10]. Where identification of this file type struggles, is that this file type is commonly found inside other file complex file types like doc (Microsoft Word), ppt (Powerpoint), or zip (compressed) files [21]. This can lead to misidentification.

### 6.1.6 Mov Vs. All

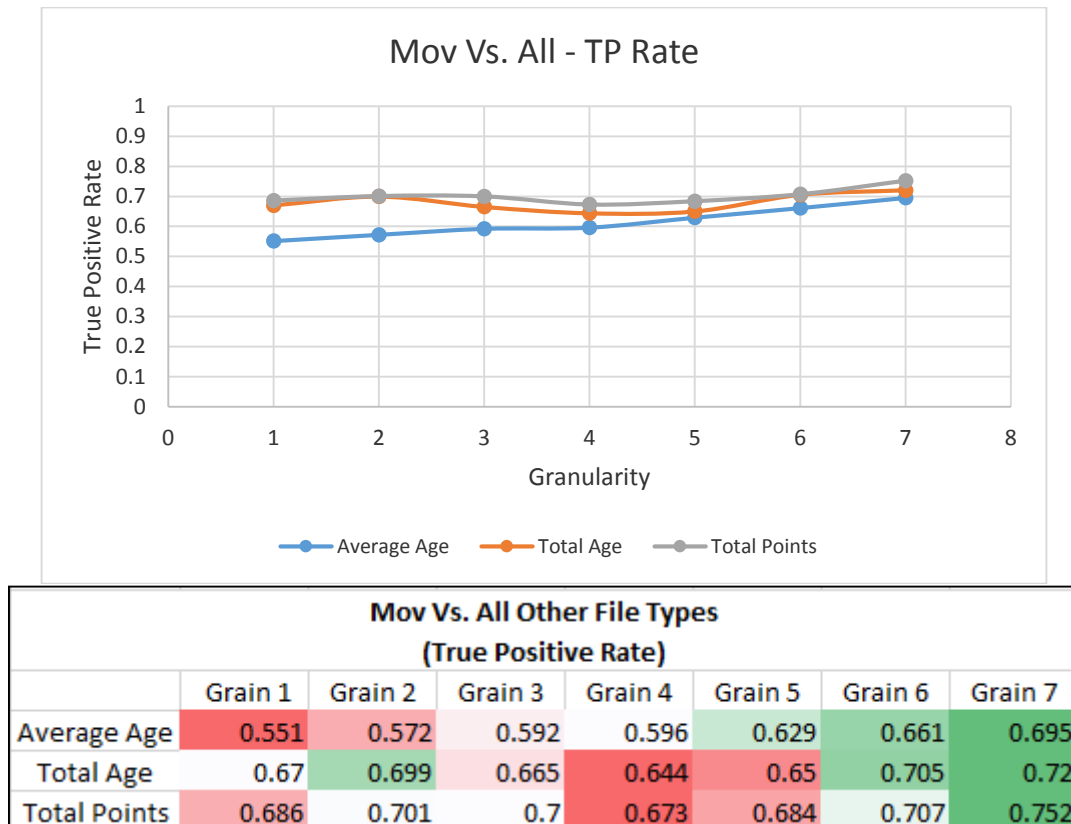
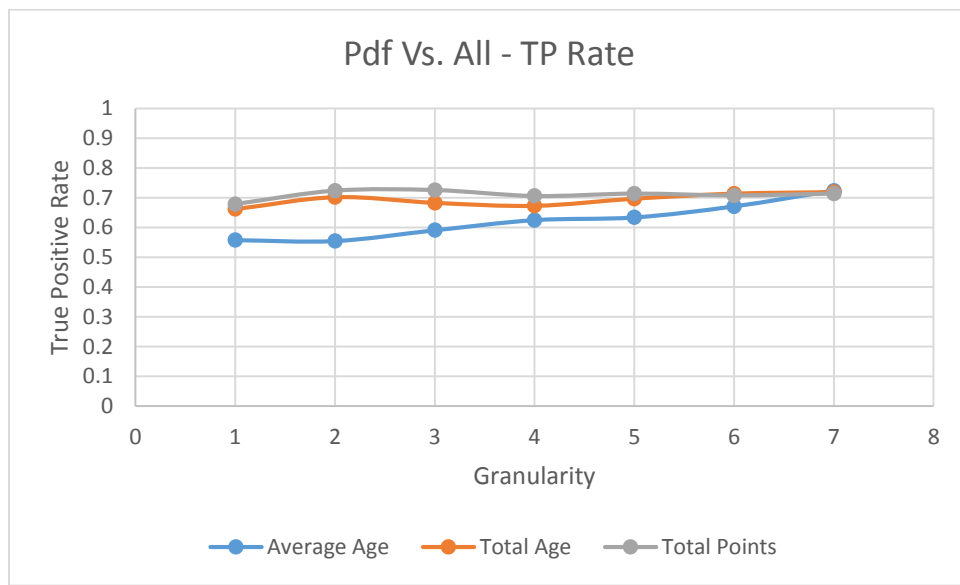


Figure 6.7 – The true positive rate of mov fragments vs. other fragment file types

The general trend of this data is that true positive rate increases as the amount of grains increase. We selected Total Points – Grain 7 for our mov model as it was both the highest true positive rate and we felt the 5% decrease in true positive rate was not enough to justify using grain 6.

**6.1.7 Pdf Vs. All**



Pdf Vs. All Other File Types (True Positive Rate)							
	Grain 1	Grain 2	Grain 3	Grain 4	Grain 5	Grain 6	Grain 7
Average Age	0.558	0.555	0.591	0.625	0.634	0.671	0.724
Total Age	0.662	0.702	0.683	0.673	0.697	0.714	0.719
Total Points	0.679	0.724	0.726	0.706	0.714	0.708	0.714

Figure 6.8 – The true positive rate of pdf fragments vs. other fragment file types

The general trend of this data is that true positive rate increases as the amount of grains increase, although grains 2 and 3 of the total points feature are notably higher than the others. The structure of a pdf file type is made more difficult to detect as, like word

documents, it can contain other objects within [1]. Thus capturing only a 512 byte section of a pdf, could very well be a portion of an embedded jpg and thus would be identified as being misclassified (as it failed to classify as a pdf) when it actually was classified correctly [21]. We selected Total Points – Grain 3 for our pdf model as it was both the highest true positive rate and a lower runtime than the higher grains.

### 6.1.8 Ppt Vs. All

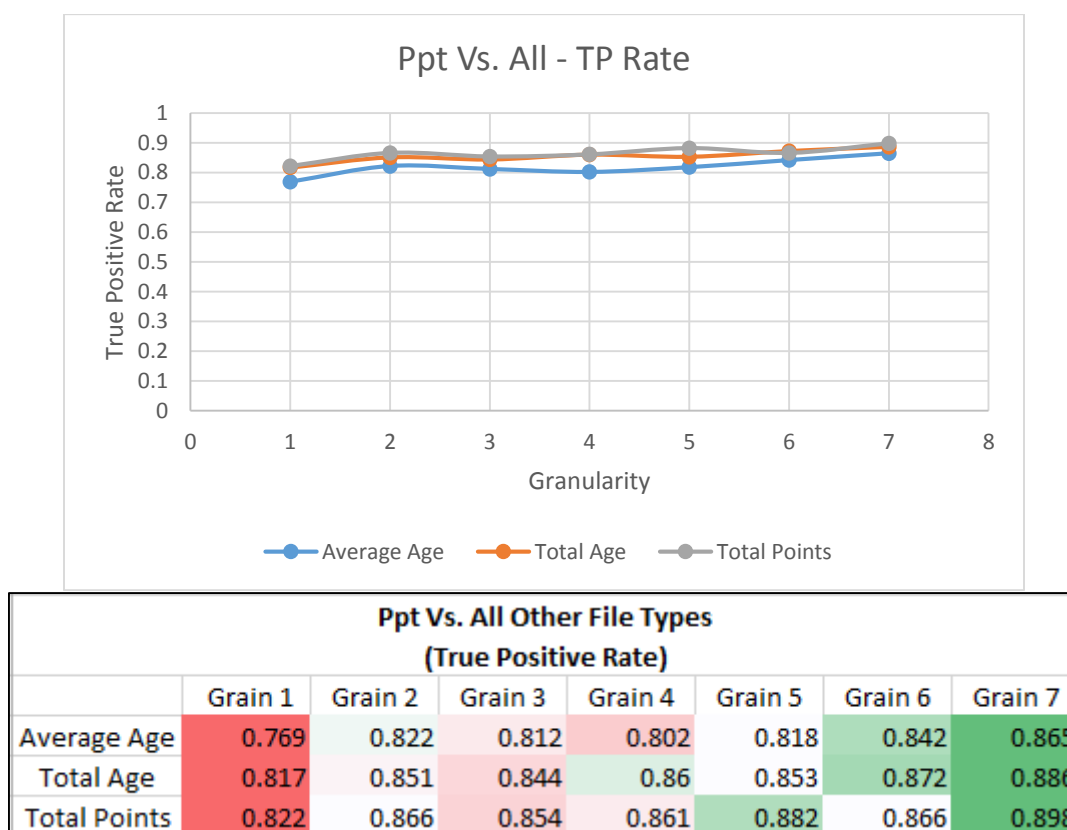
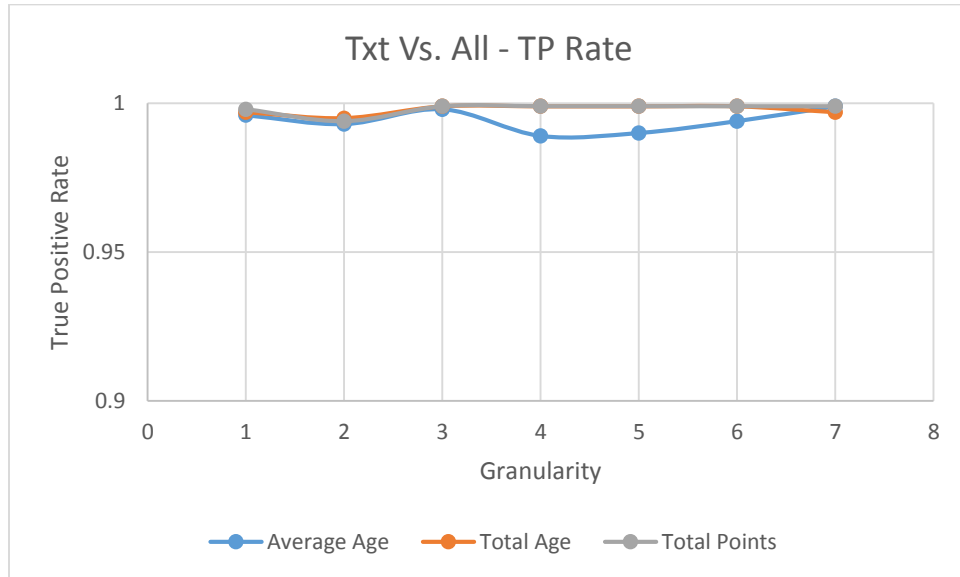


Figure 6.9 – The true positive rate of ppt fragments vs. other fragment file types

The general trend of this data is that regardless of the feature used the true positive rate forms a plateau, although the higher grains do show a slight increase. Ppt

files are also complex file types, built like smaller file systems, in that they can contain other objects within in them [21]. We selected Total Points – Grain 7 for our ppt model.

**6.1.9 Txt Vs. All**



Txt Vs. All Other File Types (True Positive Rate)							
	Grain 1	Grain 2	Grain 3	Grain 4	Grain 5	Grain 6	Grain 7
Average Age	0.996	0.993	0.998	0.989	0.99	0.994	0.999
Total Age	0.997	0.995	0.999	0.999	0.999	0.999	0.997
Total Points	0.998	0.994	0.999	0.999	0.999	0.999	0.999

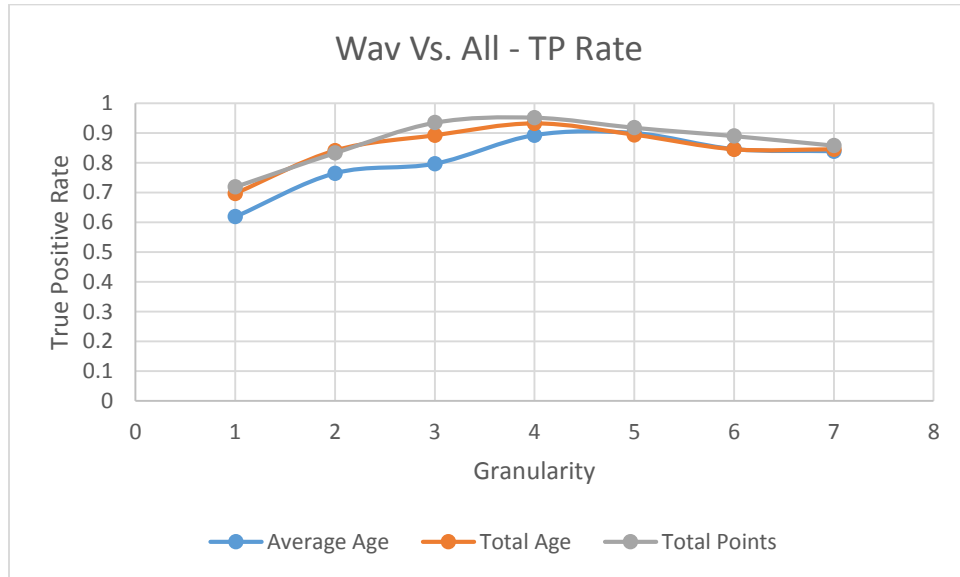
Figure 6.10 – The true positive rate of txt fragments vs. other fragment file types\*

Note\*: The TP graph has been zoomed in.

Overall the true positive rate for our txt detection was very high, so identifying noteworthy trend for this type was more difficult for this model. This is due to plain text being very basic in its structure. We noted that average sequence did the worst of the three features. We selected Total Points – Grain 4 for our txt model as the general trend

across the other models has been the higher the grain the better it will perform. Thus we selected grain 4 as it both performs well and will also take advantage of a faster runtime.

**6.1.10 Wav Vs. All**



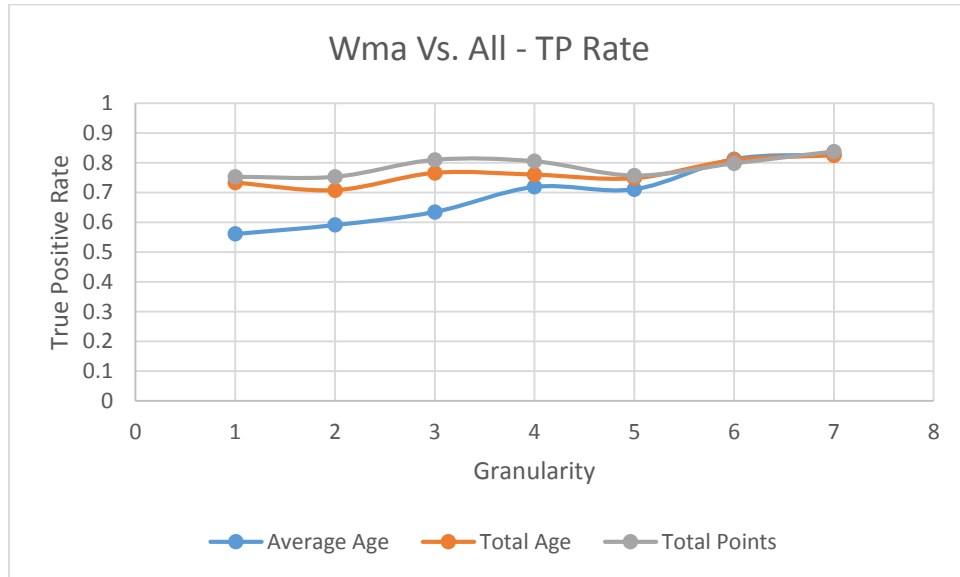
	Grain 1	Grain 2	Grain 3	Grain 4	Grain 5	Grain 6	Grain 7
Average Age	0.619	0.765	0.797	0.893	0.9	0.846	0.839
Total Age	0.697	0.841	0.892	0.932	0.894	0.845	0.845
Total Points	0.719	0.833	0.935	0.952	0.918	0.89	0.858

Figure 6.11 – The true positive rate of wav fragments vs. other fragment file types

The general trend of this data is that for all three features, the true positive rates generally fall into a parabolic curve, where the highest values fell in the middle grains. With the file types we have selected, with the exception of the zip file, wav files will not be embedded into another object amongst our 13 selected file types. The unique structure of the wav file (as shown in Figure 5.3 and Appendix B) aids VMIFF in

separating this file type from the others. We selected Total Points – Grain 4 for our wma model.

**6.1.11 Wma Vs. All**



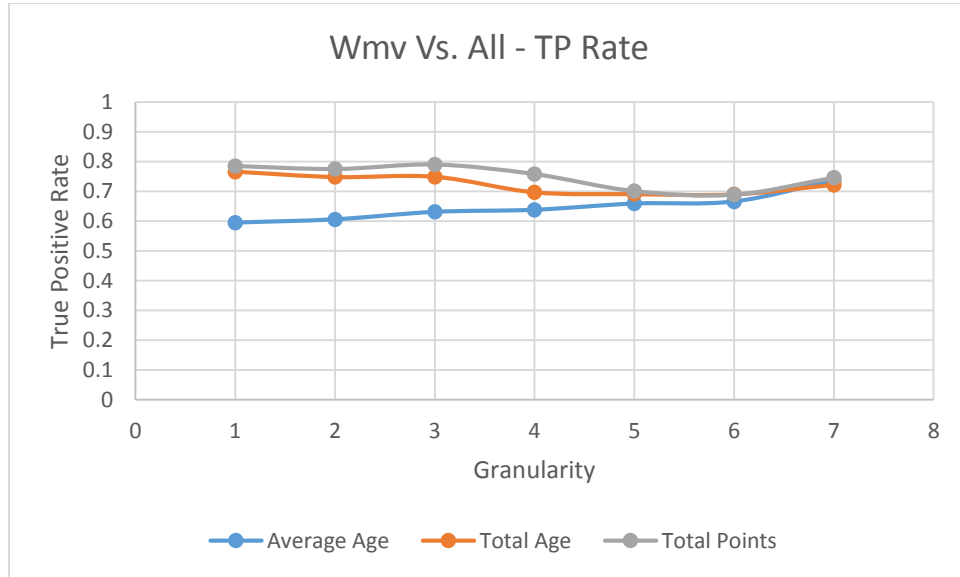
	Grain 1	Grain 2	Grain 3	Grain 4	Grain 5	Grain 6	Grain 7
Average Age	0.561	0.591	0.635	0.719	0.711	0.812	0.827
Total Age	0.733	0.708	0.766	0.76	0.748	0.809	0.825
Total Points	0.753	0.753	0.81	0.805	0.757	0.798	0.838

Figure 6.12 – The true positive rate of wma fragments vs. other fragment file types

The general trend of this data is that true positive rate increases as the amount of grains increase with a slight decrease near grain 5. Again with the exception of zip files, wma files are not likely to be embedded in the other file types. This aids in VMIFF in separating our wma file fragments from the other groups. We selected Total Points – Grain 7 for our wma model.



6.1.12 Wmv Vs. All



	Grain 1	Grain 2	Grain 3	Grain 4	Grain 5	Grain 6	Grain 7
Average Age	0.595	0.606	0.631	0.638	0.659	0.666	0.737
Total Age	0.766	0.748	0.749	0.697	0.691	0.69	0.722
Total Points	0.785	0.775	0.79	0.758	0.701	0.689	0.746

Figure 6.13 – The true positive rate of wmv fragments vs. other fragment file types

The general trend of this data is that the true positive rate is higher among the first three grains and then decreases until grain 7. Finally wmv files too will not likely be found in the selected file types we have selected. Thus our true positive rates were increased. We selected Total Points – Grain 1 for our wmv model as this was both the highest true positive rate and had the fastest runtime.

### 6.1.13 Zip Vs. All

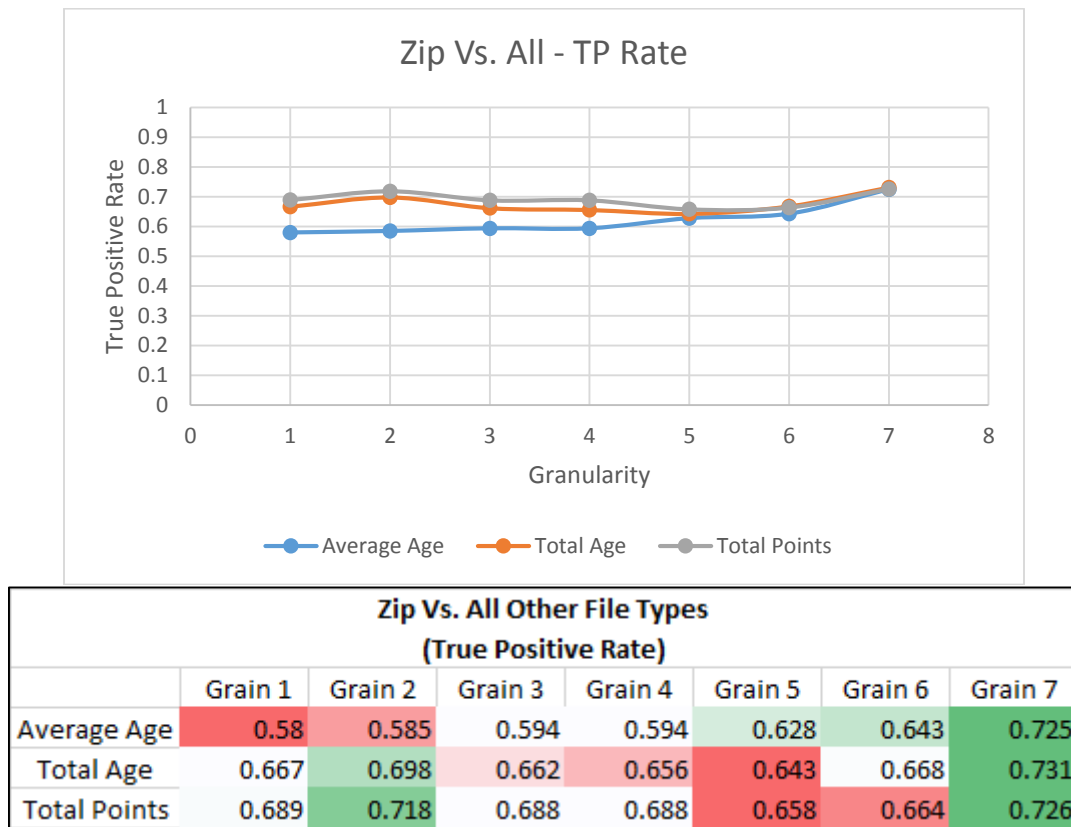


Figure 6.14 – The true positive rate of zip fragments vs. other fragment file types

The general trend of this data is that regardless of the feature used the true positive rate forms a plateau, although the higher grains do show a slight increase. Zip files too are composed like small file systems [21]. Zip files will contain other types of objects, like a jpg, inside of them [21]. This makes them a complex file type and will increase the difficulty of correctly identifying them. We selected Total Sequence – Grain 7 for our zip model as it was the highest.

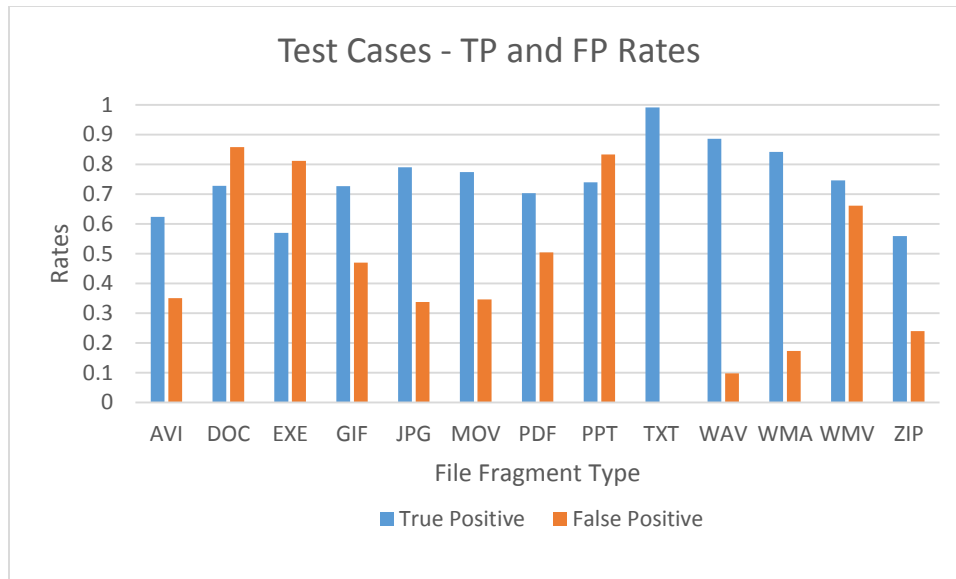
#### **6.1.14 General Trends**

Overall, from the data we collected, we noted that Average Sequence feature performed the worst of all three, while Total Points and Total Sequence had similar results. We feel that Total Points performed well as this feature better captured overall clustering of bytes in the visualization. For example wav files tend to be clustered around the central axis of the 2D graph. Thus when represented by our metric these central values would be higher in quantity than those values around the outer edge of the scatterplot. Similarly the feature Total Sequence likely performed well as it captured the overall flow (sequence) of the data.

### **6.2 Case Study – 13,000 File Fragments**

Now that we had successfully identified which features and granularity worked best to identify each file type, we built our J48 decision tree models utilizing these results. We then built a test case to run these models against.

Our test case consisted of 13,000 file fragments, 1000 of each of the 13 file types. We then ran each of our models against these file fragments to determine their effectiveness at correctly identifying the file. Our results are below (Figure 6.15).



Test Case - 13,000 file fragments		
True positive and false positive rates by type		
	True Positive (TP)	False Positive (FP)
AVI	0.624	0.351
DOC	0.728	0.858
EXE	0.57	0.812
GIF	0.727	0.47
JPG	0.791	0.338
MOV	0.774	0.346
PDF	0.703	0.505
PPT	0.74	0.833
TXT	0.992	0.001
WAV	0.886	0.098
WMA	0.842	0.173
WMV	0.746	0.661
ZIP	0.559	0.24

Figure 6.15 – The true positive and false positive rates of each file type

Our values were much like we hypothesized when building our training set. The highest false positive rates were amongst the file types which were considered to be complex in that they could contain other objects within them. This fact most likely purposed incorrect evaluation of a false positive if these complex data types contained

another of the 13 file types inside of it. We were most proud of our txt identification rate or 99.2%. Wav and Wma values were also rather high at 88.6% and 84.2% respectively with low false positive rates as well. We suggest that this is due to the three features we selected (total sequence, total points and average sequence) best being able to capture these unique forms. Additionally these three types are likely not embedded into the other file fragments, which would have decreased their likelihoods of creating a false positive.

## CHAPTER 7. DISCUSSIONS

Perhaps the largest criticism of our VMIFF tool is that the results of our models do not surpass all existing methods of file fragment identification. While this holds true, as our software stands now, we believe that by utilizing additional features and machine learning algorithms our results can still be improved upon. The modularity of our software as well as it being release in open-source format, will enable other developers and researchers to expand upon our existing methods to further enhance its usability.

The purpose of this work was to demonstrate that capturing the essence of visual data is possible and can result in a positive correlation. Thus we did not focus our efforts on determining the best machine learning algorithm for our data nor on providing an exhaustive list of potential features to measure the visualization data by. Our focus was to provide a brief demonstration of our graphing method's potential using only three features. We hope this concept will inspire others to see the value visualization of complex data can bring and to serve as a springboard for new features and new adaptations of machine learning algorithms. We hope to see our methods utilized in order to further enhance the validity and usability of visualization methods.

## CHAPTER 8. SUMMARY AND FUTURE WORKS

### 8.1 Summary

In summary, we have discussed how visualization programs can help investigators see complex information in an easy to understand format. Following this insight we developed a visualization program that can be used to visually analyze both complete files and fragments. Additionally this program will be released after April 8<sup>th</sup>, 2013, and will be available as open-source software on github under the project name VMIFF [28].

We then built a custom fragmentation program to divide up a given set of files into a specific type of fragment: whole file, header fragments, mid fragments, and footer fragments and also allowed for specification of those fragments size. We developed a graphing program which can be used to transform the visual data from the graphs into a metric which can be used in machine learning algorithms. We created a modular feature program which can allow for new features to be added without having to restructure the entire program. We demonstrated that VMIFF resulted in a positive correlation between the graphing program and the original visualization of the file fragment while using only three basic features and a 2 byte digraph.

### 8.2 Future Works

We feel that the modular aspect of VMIFF has given this tool a lot of potential. As future works we would like to see more features added to our program. A fully

developed GUI version and command line version would help to aid in the overall usability of our program. As we are releasing the program to the public as open-source, we would like to see other's insights and feedback as to this projects potential.

As to functionality, we would like to add a method to identify the order of fragments once their type has been identified. We feel that this is the next step in being able to provide a useful open-source tool to investigators. Additionally once the fragments have been identified and ordered we would like a validation tool to be created to further evaluate whether the file was correctly pieced back together. This would further reduce both the workload of the investigator and increase the automated process of recovering files.



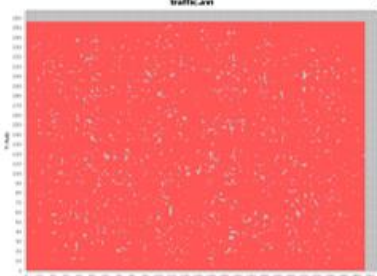
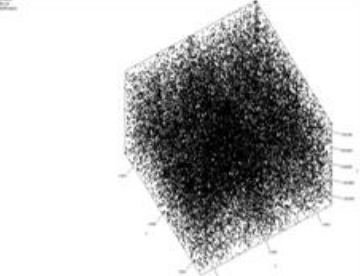
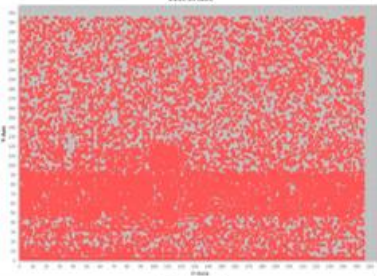
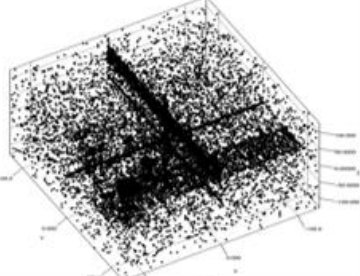
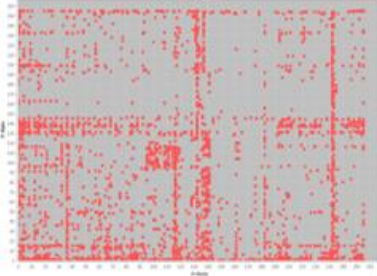
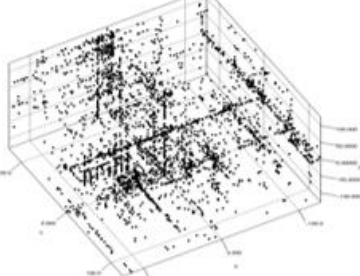
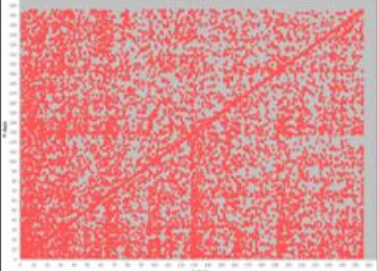
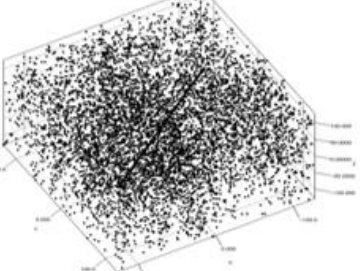
## APPENDIX A. ARFF FILE FORMAT

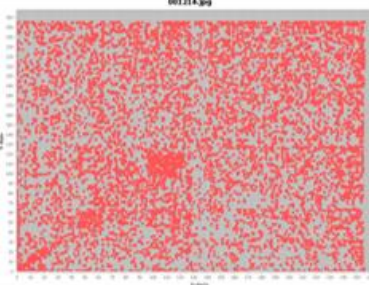
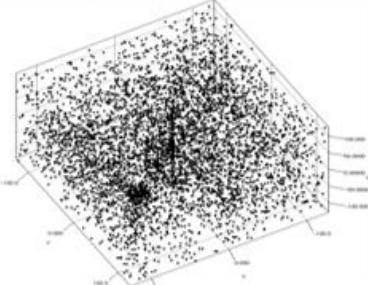

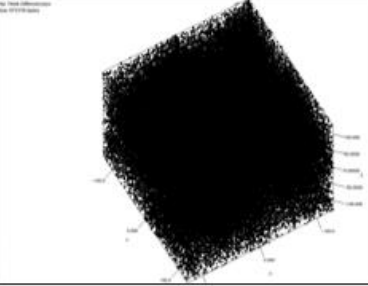
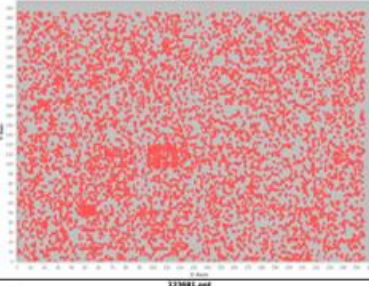
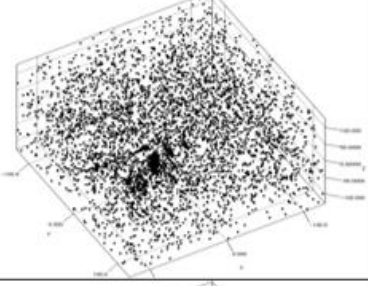
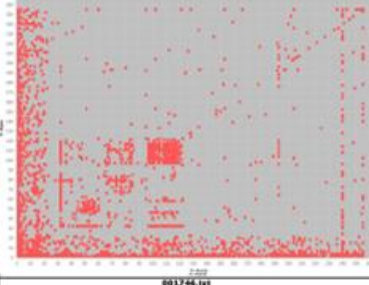
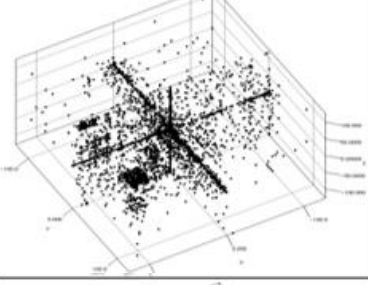
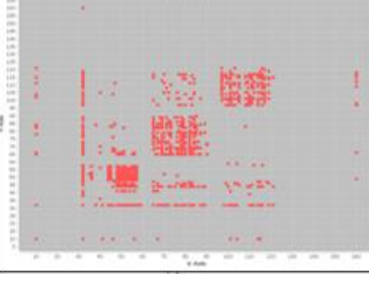
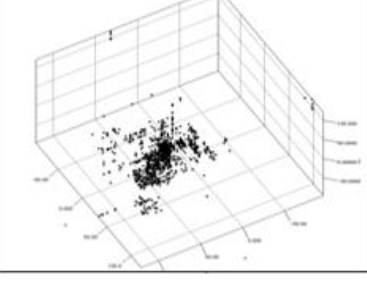
```

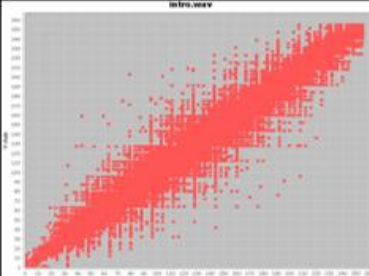
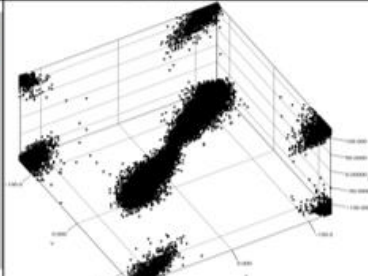
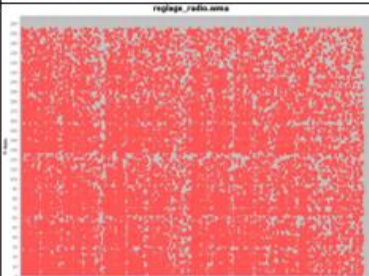
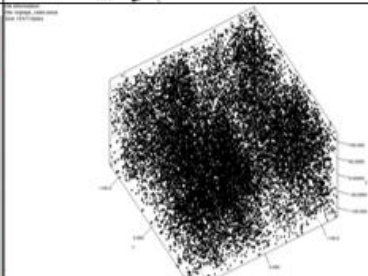
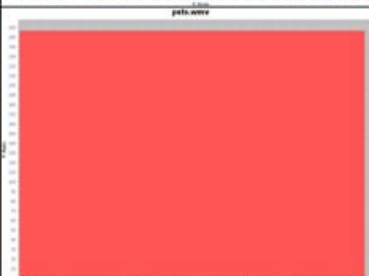
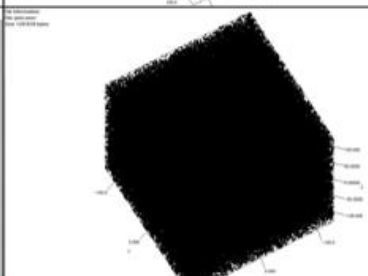
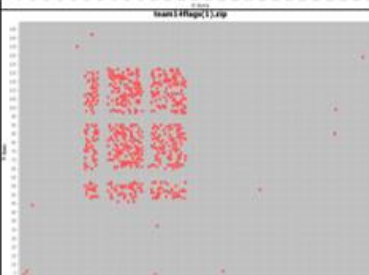
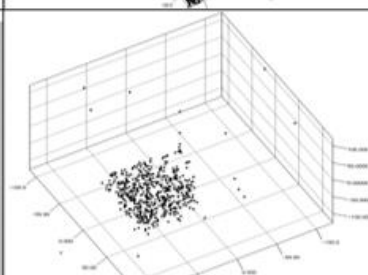
1 @RELATION SampleArffFile
2
3 @ATTRIBUTE fName string
4 @ATTRIBUTE AverageAgeFeature0 numeric
5 @ATTRIBUTE AverageAgeFeature1 numeric
6 @ATTRIBUTE AverageAgeFeature2 numeric
7 @ATTRIBUTE AverageAgeFeature3 numeric
8 @ATTRIBUTE type {1,2}
9
10 @DATA
11 Frag1,183.78343949044586,346.0,373.8181818181818,371.59183673469386,1
12 Frag10,293.51428571428573,270.2686567164179,243.46153846153845,208.44776119402985,1
13 Frag100,264.17391304347825,274.1,247.38181818181818,236.11111111111111,1
14 Frag1000,247.46666666666667,269.0181818181818,272.53125,236.28571428571428,1
15 Frag10000,280.0769230769231,242.95384615384614,258.3939393939394,225.27659574468086,1
16 Frag10001,275.6393442622951,222.42857142857142,266.63492063492066,252.78947368421052,1
17 Frag10002,244.20338983050848,235.34545454545454,265.73333333333335,268.0975609756098,1
18 Frag10003,300.51851851851853,236.91525423728814,258.03508771929825,236.8139534883721,1
19 Frag10004,256.5,287.58620689655174,241.1,236.1290322580645,1
20 Frag10005,241.22033898305085,276.96666666666664,259.45714285714286,242.80597014925374,1
21 Frag10006,235.35714285714286,254.85333333333332,255.38181818181818,270.57142857142856,1
22 Frag10007,278.8918918918919,262.0,244.58064516129033,230.3548387096774,1
23 Frag10008,252.54285714285714,267.34375,249.26153846153846,250.7017543859649,1
24 Frag10009,268.6222222222222,271.3114754098361,249.025,238.85714285714286,1
25 Frag1001,264.1463414634146,14.0,98.0,12.4,1
26 Frag10010,223.85454545454544,276.2857142857143,268.5964912280702,250.02469135802468,1
27 Frag10011,250.72413793103448,259.8787878787879,258.39344262295083,251.04225352112675,1
28 Frag10012,228.90322580645162,240.0327868852459,288.4727272727273,263.84615384615387,1
29 Frag10013,262.03125,250.62962962962962,260.4415584415584,244.62295081967213,1
30 Frag10014,275.0153846153846,245.73333333333332,244.94117647058823,254.03174603174602,1
31 Frag1226,257.8425925925926,263.05263157894734,226.4,60.0,2
32 Frag12260,251.4742268041237,239.8918918918919,307.1764705882353,299.5,2
33 Frag12261,249.1356783919598,277.65,268.14285714285717,280.6666666666667,2
34 Frag12262,262.37894736842105,260.7368421052632,232.2,109.5,2
35 Frag12263,261.7857142857143,221.94117647058823,258.55555555555554,221.25,2
36 Frag12264,254.26666666666668,227.1578947368421,283.06666666666666,352.5,2
37 Frag12265,251.70408163265307,275.1707317073171,254.875,195.33333333333334,2
38 Frag12266,240.87046632124353,300.85714285714283,236.0,386.0,2
39 Frag12267,257.5049504950495,262.1818181818182,216.0,235.0,2
40 Frag12268,259.65829145728645,234.10256410256412,250.5,235.0,2
41 Frag12269,265.65405405405403,225.25,273.5,80.0,2
42 Frag1227,256.6574074074074,180.14285714285714,279.1818181818182,294.5,2
43 Frag12270,265.8181818181818,198.64864864864865,251.78947368421052,257.0,2
44 Frag12271,252.64615384615385,258.1463414634146,279.11111111111111,203.0,2
45 Frag12272,247.34299516908212,311.54545454545456,259.6923076923077,474.0,2
46 Frag12273,258.58378378378376,272.90909090909093,251.1578947368421,82.75,2
47 Frag12274,258.7177033492823,228.06451612903226,260.42857142857144,246.0,2
48 Frag12275,251.45077720207254,239.31578947368422,291.6470588235294,337.25,2
49 Frag12276,251.47738693467338,269.8974358974359,250.66666666666666,316.6666666666667,2
50 Frag12277,259.7578947368421,254.4102564102564,274.2105263157895,99.25,2

```

## APPENDIX B. 2D AND 3D VISUALIZATIONS OF FILES

Visualization of File Types		
File Extension	2D Graph	3D Graph
AVI (traffic.avi)		
DOC (006767.doc)		
EXE (genmkvpwd.exe)		
GIF (008157.gif)		

Visualization of File Types		
File Extension	2D Graph	3D Graph
JPG (001214.jpg)		
MOV (Think Differently.mov)		
PDF (218835.pdf)		
PPT (223681.ppt)		
TXT (232073.txt)		

Visualization of File Types		
File Extension	2D Graph	3D Graph
WAV (intro.wav)		
WMA (reglage_radio.wma)		
WMV (pets.wmv)		
ZIP (team14flags(1).zip)		

## BIBLIOGRAPHY

- [1] – Adobe Systems Incorporated. *Document management — Portable document format — Part 1: PDF 1.7*. 2008. [https://www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000\\_2008.pdf](https://www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf)
- [2] *Cantor.Dust*. Battelle Memorial Institute, 2012. <https://sites.google.com/site/xxcantorxdustxx/home>.
- [3] Conti, Gregory; Bratus, Sergey. "Voyage of a Reverse." BlackHat. Las Vegas, NV. 29 July 2010. Web. 2013. [http://media.blackhat.com/bh-us-10/presentations/Bratus\\_Conti/BlackHat-USA-2010-Bratus-Conti-Voyage-of-a-Reverser-slides.pdf](http://media.blackhat.com/bh-us-10/presentations/Bratus_Conti/BlackHat-USA-2010-Bratus-Conti-Voyage-of-a-Reverser-slides.pdf).
- [4] Conti, Gregory; Bratus, Sergey; Shubina, Anna; Lichtenberg, Andrew; Sangster, Benjamin; Supan, Matthew. "A Visual Study of Primitive Binary Fragment Types. 2010.
- [5] – Conti, Gregory; Bratus, Sergey; Shubina, Anna; Lichtenberg, Andrew; Sangster, Benjamin; Supan, Matthew; Lichtenberg, Andrew ; Perez-Aleman, Robert. Automated mapping of large binary objects using primitive fragment type classification, *Digital Investigation*, Volume 7, Supplement, August 2010, Pages S3-S12, <http://www.sciencedirect.com/science/article/pii/S1742287610000290>
- [6] Conti, Gregory; Dean, Erik; Sinda, Matthew; Sangster, Benjamin. 2008. Visual Reverse Engineering of Binary and Data Files. In *Proceedings of the 5th international workshop on Visualization for Computer Security (VizSec '08)*, John R. Goodall, Gregory Conti, and Kwan-Liu Ma (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-17. [http://dx.doi.org/10.1007/978-3-540-85933-8\\_1](http://dx.doi.org/10.1007/978-3-540-85933-8_1).
- [7] *Digital Corpora - Govdocs I*. Garfinkel, Farrell, Rousev and Dinolt, *Bringing Science to Digital Forensics with Standardized Forensic Corpora*, DFRWS 2009 13 Nov. 2012. <http://digitalcorpora.org/corpora/files>.
- [8] Domas, Christopher. "The Future of RE: Dynamic Binary Visualization." DerbyCon. Louisville, KY. . Web. 28 Sept. 2012. <http://www.irongeek.com/i.php?page=videos/derbycon2/4-2-1-christopher-domas-the-future-of-re-dynamic-binary-visulization>.
- [9] Drazin, Sam, and Matt Montag. "Decision Tree Analysis using Weka Machine Learning – Project II." *University of Miami*. <http://www.samdrazin.com/classes/een548/project2report.pdf>.
- [10] – Erbacher, R.F.; Mulholland, J., "Identification and Localization of Data Types within Large-Scale File Systems," *Systematic Approaches to Digital Forensic Engineering*, 2007. *SADFE 2007. Second International Workshop on* , vol., no.,

pp.55,70, 10-12 April 2007 <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4155350&isnumber=4155338>

[11] Fiedorowicz, Zbigniew. *Space Filling Curve - Hilbert Curve Graphs*. Ohio State University. <http://www.math.osu.edu/~fiedorowicz.1/math655/Peano.html>.

[12] *FileInfo.com - The Central File Extension Registry*. 2013. <http://www.fileinfo.com/>.

[13] Garfinkel, Simson L. "Carving contiguous and fragmented files with fast object validation." Elsevier Ltd (2007): S12-S2. 18 Nov. 2011.

[14] Hall, Mark ; Frank, Eibe; Holmes, Geoffrey; Pfahringer, Bernhard; Reutemann, Peter; Witten H., Ian. (2009); *The WEKA Data Mining Software: An Update*; SIGKDD Explorations, Volume 11, Issue 1.

[15] Kenney, Brad. "Every Millisecond Counts." *IndustryWeek (IW)* 1 Apr. (2008). Web. 26 Nov. 2011. [http://www.industryweek.com/articles/every\\_millisecond\\_counts\\_15932.aspx](http://www.industryweek.com/articles/every_millisecond_counts_15932.aspx).

[16] Memon N, Pal A. Automated reassembly of file fragmented images using greedy algorithms. *IEEE Trans Image Process*. 2006 Feb; 15(2):385-93.

[17] Nance, K.; Armstrong, H.; Armstrong, C., "Digital Forensics: Defining an Education Agenda," *System Sciences (HICSS)*, 2010 43<sup>rd</sup> Hawaii International Conference on, vol., no., pp.1,10, 5-8 Jan. 2010. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5428493&isnumber=5428274>.

[18] Object Refinery Limited. *JFreeChart*. Object Refinery Limited, 2012. <http://www.jfree.org/jfreechart/>.

[19] Pal, A.; Memon, N., "The evolution of file carving," *Signal Processing Magazine, IEEE*, vol.26, no.2, pp.59,71, March 2009. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4806206&isnumber=4806187>.

[20] Pernollet, Martin. *JZY3D*. 2013. <http://www.jzy3d.org/index.php>.

[21] Roussev, V.; Garfinkel, S.L., "File Fragment Classification-The Case for Specialized Approaches," *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE '09. Fourth International IEEE Workshop on*, vol., no., pp.3,14, 21-21 May 2009. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5341545&isnumber=5341544>.

[22] Suhyung Jo; Do-Won Hong, “The study of text extraction for forensic data,” *Networked Computing and Advanced Information Management (NCM), 2011 7<sup>th</sup> International Conference on*, vol., no., pp.186,189, 21-23 June 2011.  
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5967542&isnumber=5967504>.

[23] Uniblue Systems Ltd. *FILExt - The File Extension Source*. Uniblue Systems Ltd, 2013. <http://filext.com/>.

[24] United States Census Bureau – *Households with a Computer and Internet Use: 1984 to 2009*. October 2009 <https://www.census.gov/hhes/computer/files/Appendix-TableA.xls>.

[25] United States Census Bureau. *Presence and Type of Computer for Households, by Selected Householder Characteristics 2010*. <https://www.census.gov/hhes/computer/files/2010/table1C.csv>.

[26] Venkatraman, Dheera. *FooPlot | Online Graphing Calculator*. 2013.  
<http://www.fooplot.com/>.

[27] – Vitelaru, Davide. *Digital Janitor*. 2011. <http://davidevitelaru.com/software/digital-janitor/>.

[28] VMIFF Github Repository - <https://github.com/hartell/VMIFF.git>.

[29] Wood, Roger. “Future hard disk drive systems.” *Journal of magnetism and magnetic materials* 321.6 (2009): 555-561.