2018-06

# CONSTRUCTING SOCIAL NETWORKS AND CLASSIFYING EMAIL ADDRESSES FROM RAW FORENSIC IMAGES

Goodwin, Justin; Ward, Erin C.

Monterey, CA; Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**CONSTRUCTING SOCIAL NETWORKS AND CLASSIFYING EMAIL ADDRESSES FROM RAW FORENSIC IMAGES**

by

Erin C. Ward and Justin Goodwin

June 2018

| | |
|---|---|
| Thesis Advisor: | Michael R. McCarrin |
| Co-Advisors: | Marcus S. Stefanou |
| | Ralucca Gera |

**Approved for public release. Distribution is unlimited.**

THIS PAGE INTENTIONALLY LEFT BLANK

| REPORT DOCUMENTATION PAGE | | | | *Form Approved OMB No. 0704-0188* |
|---|---|---|---|---|



| | | |
|---|---|---|
| **1. AGENCY USE ONLY** ***(Leave blank)*** | **2. REPORT DATE** June 2018 | **3. REPORT TYPE AND DATES COVERED** Master's thesis |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** CONSTRUCTING SOCIAL NETWORKS AND CLASSIFYING EMAIL ADDRESSES FROM RAW FORENSIC IMAGES | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Erin C. Ward and Justin Goodwin | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(E**S) N/A | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER** |

**11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| | |
|---|---|
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release. Distribution is unlimited. | **12b. DISTRIBUTION CODE** A |

**13. ABSTRACT (maximum 200 words)**

The ability to find email addresses on digital storage media and deduce the relationships between them is critical for the success of many law enforcement and intelligence collection activities. Currently, building these social networks requires manually processing forensic images of acquired digital media. We conduct an experiment using readily available extraction and visualization tools along with a new algorithm that constructs networks based on the byte-offset proximity between digital artifacts. The main objective of this study is to test this new algorithm and refine techniques for classification with a goal of automating steps in the process of constructing social networks. To achieve this, we build an 11 terabyte dataset of drive images, construct networks from them, and assign these networks to the categories "useful" or "not useful" according to whether we believe them to contain information relevant to the actual social network of the device owner. We then interview device owners to determine ground truth, which we use to evaluate our analysis. We succeed in correctly categorizing networks with a recall of 0.9166, precision of 0.6316 and F-score of 0.7643. Our results show that our algorithm is successful in outputting data useful for the construction of the user's social networks.

| | |
|---|---|
| **14. SUBJECT TERMS** digital forensics, social networks, graph theory | **15. NUMBER OF PAGES** **117** |
| | **16. PRICE CODE** |

| | | | |
|---|---|---|---|
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UU |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release. Distribution is unlimited.**

**CONSTRUCTING SOCIAL NETWORKS AND CLASSIFYING EMAIL ADDRESSES FROM RAW FORENSIC IMAGES**

Erin C. Ward
Chief Warrant Officer 3, United States Army
BS, University of Phoenix, 2012

Justin Goodwin
Lieutenant, United States Navy
BS, University of Kansas, 2009

Submitted in partial fulfillment of the
requirements for the degrees of

**MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS**

and

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
**June 2018**

Approved by: Michael R. McCarrin
Advisor

Marcus S. Stefanou
Co-Advisor

Ralucca Gera
Co-Advisor

Dan C. Boger
Chair, Department of Information Sciences

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

The ability to find email addresses on digital storage media and deduce the relationships between them is critical for the success of many law enforcement and intelligence collection activities. Currently, building these social networks requires manually processing forensic images of acquired digital media. We conduct an experiment using readily available extraction and visualization tools along with a new algorithm that constructs networks based on the byte-offset proximity between digital artifacts. The main objective of this study is to test this new algorithm and refine techniques for classification with a goal of automating steps in the process of constructing social networks. To achieve this, we build an 11 terabyte dataset of drive images, construct networks from them, and assign these networks to the categories "useful" or "not useful" according to whether we believe them to contain information relevant to the actual social network of the device owner. We then interview device owners to determine ground truth, which we use to evaluate our analysis. We succeed in correctly categorizing networks with a recall of 0.9166, precision of 0.6316 and F-score of 0.7643. Our results show that our algorithm is successful in outputting data useful for the construction of the user's social networks.

THIS PAGE INTENTIONALLY LEFT BLANK

# Table of Contents

# List of Figures

# List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

# List of Acronyms and Abbreviations

| | |
|---|---|
| **APFS** | Apple File System |
| **CSV** | Comma Separated Values |
| **DoD** | Department of Defense |
| **FAT** | File Allocation Table |
| **FTK** | Forensic Toolkit |
| **GEXF** | Graph Exchange XML Format |
| **GFP** | Graph FingerPrint |
| **GFPC** | Graph FingerPrint Comparison |
| **IMAP** | Internet Message Access Protocol |
| **IRB** | Institutional Review Board |
| **JPEG** | Joint Photographic Experts Group |
| **KIC** | Kodak Compressed Image |
| **LBA** | Logical Block Address |
| **NAND** | Negative AND |
| **NPS** | Naval Postgraduate School |
| **NTFS** | New Technology File System |
| **PDF** | Portable Document Format |
| **PNG** | Portable Network Graphics |
| **POP3** | Post Office Protocol 3 |

| | |
|---|---|
| **RAR** | Roshal Archive |
| **RBF** | Radial Basis Function |
| **ROC** | Receiver Operating Characteristic |
| **RPM** | Revolutions Per Minute |
| **SSD** | Solid State Drive |
| **SMTP** | Simple Mail Transfer Protocol |
| **SVM** | Support Vector Machine |
| **TSK** | The Sleuth Kit |
| **USB** | Universal Serial Bus |
| **USG** | United States government |
| **URL** | Uniform Resource Locator |
| **USN** | U.S. Navy |
| **VM** | Virtual Machine |
| **XML** | Extensible Markup Language |

# Acknowledgments

We would like to thank our families: Andrew Ward, Logan Ward, Leslie Higgen, Dan Moore, and Desiree Moore. Their love and support is what got us to this point in our careers. We would also like to thank our thesis advisors: Michael McCarrin, Dr. Ralucca Gera, and Dr. Marcus Stefanou. We would not have been able to do this without their support.

Michael, your expertise in this field and mentorship has helped us shape our research and guide us through the thesis process. Ralucca, you always provided quick and relevant feedback as well as positive encouragement and guidance. Marcus, thank you for being committed to student success and growth. Teaching is not a job for you, it is a passion and we truly appreciated learning from you. The level of dedication and effort by all of our advisors is truly inspiring.

We would also like to thank the participants who volunteered for our research. Thank you for taking time out of your busy schedules to help us reach our goals and for trusting us.

Our tours at NPS were fraught with family tragedies. We would like to thank the professors and administrative staff for their understanding and assistance during those difficult times.

Lastly, we would like to thank our friends and fellow classmates for making our time at NPS both educational and memorable. We have made friendships that we will cherish.

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 1: Introduction

With the increasing prevalence of digital technologies, more aspects of a person's daily life are captured in digital form. This digital data provides a trove of potentially useful evidence for investigators [1]. Forensic analysts from both law enforcement and intelligence communities face the task of sifting through this vast amount of digital data and uncovering information relevant to their investigations [1]. The quantity of data requiring analysis can cause delays of weeks and even months at data forensics laboratories [1]. Many times, a key element to an investigation is the social network of the person being investigated—that is the network of friends, coworkers, and other associates that the individual communicates with.

Unfortunately, reconstructing an individual's social network from digital evidence is labor intensive. A forensic analyst must examine a large amount of data and determine the relevance of each piece of information or artifact. One method for potentially accelerating this process is bulk analysis. Bulk analysis is a technique used to extract select forensic artifacts from secondary storage devices operating directly at the disk sector level and searching for artifacts in the raw binary data, without reference to higher-level structures such as files or file system metadata [2]. The advantage of using bulk analysis is increased speed and thoroughness, but the downside can be a high percentage of useless or unwanted artifacts. Since most extracted artifacts are not useful to the analyst, an automated process that distinguishes between useful and useless digital forensic artifacts will save time during analysis. Once artifacts are extracted, the analyst must use them to deduce the target's social network—a process which requires considerable time and effort. In order to automate this task, our research focuses one type of digital forensic artifact: the email address.

Email addresses can be of great significance when reconstructing a participant's social network. They provide information about who the drive owner is communicating with and the frequency of the communications. Unfortunately, not all email addresses are helpful. There are many ways an email address or a string of characters that resembles an email address can be present on the drive and have no correlation to the communications of the user. For example, software installations and cached websites contain email addresses that are unrelated to the user's personal contacts or associates. Even when a forensic analyst is specifically examining email artifacts, the dataset can be large and contain an abundance of email addresses that are not in the user's social network.

## 1.1 Motivation

The motivation for our research is the reduction of time that a forensic analyst has to spend manually examining data to find email addresses that directly contribute to developing a user's social network. Building an accurate social network from information on a user's drive is a multistage process that requires human interaction on almost every stage. The intent of our research is to improve current methods used to process data and to reduce useless data, referred to as "noise," that a forensic analyst needs to examine. To achieve this goal, we are testing a new algorithm for identifying connections between email addresses found on the participant's drive. The research question this thesis seeks to answer is the following:

*Does our new algorithm for constructing social networks improve our ability to correctly model user communication patterns?*

Previous research has shown the physical proximity of email addresses on secondary storage media is sufficient to establish possible social networks on the participant's device [3], [4]. Our work continues the previous research conducted by Green [3] and Allen [4]. They made advancements toward reducing noise captured during artifact extraction and toward the ultimate goal of automatically constructing the participant's social networks. The algorithm they tested defines a fixed byte window and constructs graphs with links between email addresses that fall within that window. Green's analysis indicates that using measures of centrality assist in identifying important nodes and close associates. Allen was able to extract summary attributes from the same graphs of email addresses to classify them into the categories "useful" and "not useful." Although the sample size was smaller than desired, their results warranted further research on a larger data set.

Due to a limitation in the version of friend.py Green and Allen used, email addresses appearing at the end of long sequences were linked to each other but not to addresses at the beginning of the sequence. The network was distorted in these cases and addresses in the same email headers were not equally correlated. The new algorithm we are testing will attempt to resolve this problem by treating the window as a "timeout" value that refreshes every time it locates a new email address. The algorithm will link all email addresses within the window and will continue to link the email addresses until the timeout ends. This produces a more accurate graph.

## 1.2 Contributions

To evaluate the effectiveness of our algorithm, we collected a new, larger data set and conducted interviews to determine ground truth for each drive. This created an 11 terabyte corpus of drives with ground truth. In our experiment, we attempted to model the participant's communication patterns using Gephi to visualize the output from bulk_extractor and our new algorithm. For each drive collected, we analyzed the resulting graphs using concepts such as node count, edge count, modularity class, k-core, and degree. We attempted to determine the email address of the drive owner as well as whether each subgraph contained useful information about the drive owner's social network. We used the ground truth obtained during the interviews to validate our assessment of the subgraphs and determine the effectiveness of our process. We were able to successfully identify subgraphs that contained the participant's social network with a recall of 0.9677, precision of 0.6316, and F-score of 0.7643.

During our research, we identified a bug in bulk_extractor. Correcting this will reduce noise in the email.txt feature file by 24%. In addition, we found that some of the subgraphs contained a small amount of useful data buried in a large amount of noise, and we developed a methodology for separating out these "hybrid" graphs using community detection. Finally we identified several similar subgraphs that were found on different drives. We established identifying characteristics of these common graphs and created a taxonomy of graphs that can be used for future work building a classifier.

## 1.3 Thesis Structure

Our thesis is presented in the following order. Chapter 2 presents technical background information on forensic and visualization techniques as well as currently used tools. Chapter 3 describes related work in the areas of social network analysis, email analysis, and bulk data analysis. Chapter 4 presents our methodology and experiment design for creating social network graphs. Chapter 5 describes our analysis and results from our experiment. In Chapter 6 we present our conclusion and propose areas for future work.

# CHAPTER 2:
# Technical Background

This chapter introduces the concepts and tools that we used throughout our experiment. The concepts covered in this chapter come from digital forensics, network science, and graph theory.

We begin by providing a description of how and why email addresses are stored on secondary storage devices. Several tools we use originate from work in digital forensics. We explain the functionality of these tools and how we used them to process the data. We then proceed to network science and graph theory sections: these provide useful definitions and basic information that aids in understanding graphing and visualization of the social networks. Finally, we describe the measures of performance we used to evaluate the success of our experiment.

## 2.1 Email Addresses on Secondary Storage

Email addresses show up on secondary storage for a variety of reasons. We outline several possible sources of email addresses, including: web browsers, application installation, Microsoft Word documents containing email addresses, and strings that appear to have the same format as email addresses, such as references to portable network graphics (PNG) file names in applications. For example, web browsers cache websites under user's profiles and often an email address is used to log into a website which is stored in a file. In Figure 2.1, we follow the path from user level to the block device to show how email addresses are stored with different applications. In this section, we will describe properties of email addresses, the operating system file structure, and secondary storage devices which explain how we found the email addresses on secondary storage.

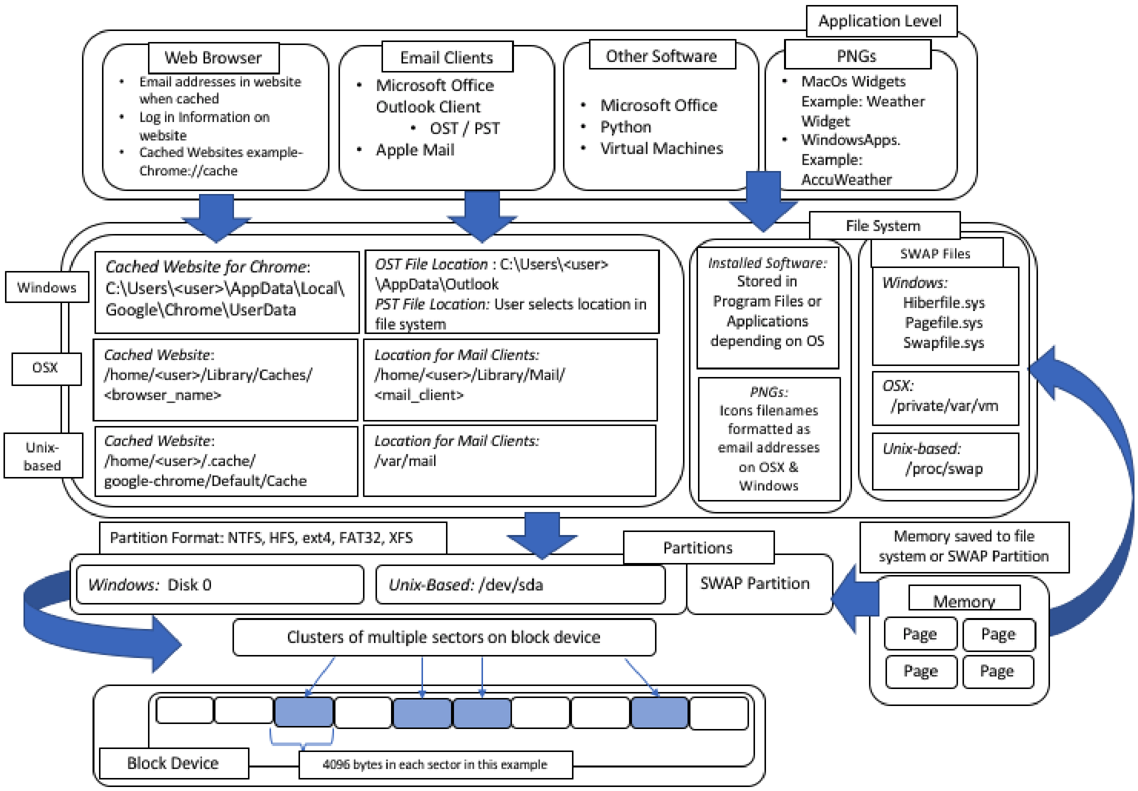


Figure 2.1. Example of ways email addresses are stored on secondary storage devices by applications. The figure describes Windows, Mac OS X, and Unix-based operating systems.

## 2.2 Email Properties

Understanding the various ways email addresses are used along with the associated protocols is important for explaining why a given email address is present on a secondary storage device. Not every email address found on a drive belongs to a person in the drive owner's social network. Website caches, software installation, and virtual machines store email addresses in files which are stored on secondary storage. Email addresses from these sources offer little or no value when attempting to recreate the owner's social network.

The two main ways a user might access their email are through an email client that stores email content to disk for offline editing or through a web based email service. Each of these methods interact with the computer's storage device in different ways. The primary difference is that a web based email client requires the user to connect through the Internet to access an email or compose a new email. In the case of local email client, a constant internet connection is not required to read messages or compose new email.

**Email Protocols**

Email protocols contain two components, one to retrieve incoming mail and one to send outgoing mail. There are different protocols used with email servers to transmit data to the client. The type of protocol used can affect how much data is stored on the local hard drive. Therefore, the email protocols have an impact on our research.

Post Office Protocol 3 (POP3) allows clients to download email data from the server and store it on the local machine. The individual mail client determines how to store the data on the local machine. With the POP3 protocol, only new email is stored on the server and once it is downloaded by the client, the server usually does not maintain any record of the email [5]. Other email functions, such as email organization and contact lists, are handled by the client. POP3 is most appropriate for clients that lack or do not have a persistent Internet connection.

Internet Message Access Protocol (IMAP) acts more like a remote file server because it allows clients to download new email, as well as organize and store data on the server. When the connection is made to the server, the client synchronizes its data with the server [6]. For an email to be read, the data must be downloaded and stored on the client's device [6].

Simple Mail Transport Protocol (SMTP) is a protocol used to send new mail from the client to a mail server where they will be stored. It is typically used with either POP3 or IMAP to complete the email exchange process.

### 2.2.1 Client Based Email

Client based email is a piece of software that is installed on the user's local computer and creates a user interface for reading, writing, and other email functions. The entire user interface as well as the email, contact lists, and calendars are stored on the local drive. This allows the user to read, write, and manipulate data while offline and provide a local back up for information on the server. The client uses standard protocols, described previously, to retrieve and send email to and from the mail server when an Internet connection is available.

Three commonly used client based email programs are Outlook, Thunderbird, and OS X Mail.

### 2.2.2 Web-based Email

Web-based email applications are hosted on remote servers. This type of email interface is accessed through a web browser. There is no required download or installation on your local computer to use web-based email. Some commonly used web-based email providers are Gmail, Yahoo, and Outlook. Many client-based email providers such as Outlook also provide web-based access. Typically, most of the email data is stored on the remote mail servers. However, there are some instances where email data can be stored on the local drive.

In the case of Microsoft Outlook, the user can download files called Personal Storage Table (.pst) or Offline Outlook Data File (.ost) as a means of saving their email to their local drive. Additionally, if a process (web-based email) is running on the computer it will use local memory. If another process requires that memory, data from the email process will be swapped to the drive. This data likely contains email addresses. In addition, website user names such as email addresses are commonly stored by web browsers.

## 2.3 Operating System's File Structure

When conducting digital forensics on a raw drive image an analyst does not always have the luxury of knowing what operating system is installed. The type of operating system is useful to know because each operating system uses a unique method to store and retrieve data from a storage device. We are using tools that bypass the file system and are able to extract data regardless of the operating system installed. As shown in Figure 2.1, the file system is a part of the process of storing email addresses on secondary storage. The operating system's file structure or file system is useful for displaying the files, folders, and directories to the user. Files and folders become useful during our experiment when we need to know where email addresses are extracted from on the drive. Our data set consists primarily of Windows operating systems and Mac OS X operating systems.

**File Storage and Deletion**

The file system stores key information about the data that makes up each file stored on the drive. Along with other information, it contains the name, size, and exact location where the data associated with the file is stored on the drive. The data for a file could be stored together or in consecutive sectors. However, practical application sometimes requires data to be stored in several fragments on the drive, a condition known as "file fragmentation." The file system keeps track of where the sectors are that contain data for each file.

In most file systems, when a file is deleted by a user, the data is not actually erased from the drive. The reference to the file is simply removed from the file system and that space on the storage drive is now available to hold new data. All of the data that was previously stored in that location remains and can be read through data forensics tools.

**Windows File Systems**

Microsoft Windows uses two main file systems, FAT (File Allocation Table) and NTFS (New Technology File System). FAT is the older, legacy file system which uses a table of files and a linked list to connect the sectors. NTFS uses a master file table and stores many additional file properties. It uses a wider range of values as references so it can support high capacity storage devices.

**Mac OS File Systems**

The Mac operating system is based on Unix and has many similarities. The Mac operating system a uses Hierarchical File System (HFS+) The basic function is the same; however the Catalog file, rather than the Master File Table tracks which sectors are used,and the Attribute File stores the file attributes. In 2017, Apple introduced a new file system called Apple File System (APFS) which is optimized for solid state drives [7].

## 2.4 Properties of Storage Devices

Data storage devices have evolved significantly over the last hundred years. The size of devices has decreased while the storage capacity has increased drastically. The modern devices we use today find their origins in punch cards, paper tape, and magnetic tape. Each of these technologies changed the current thought about data storage and was a dominant mechanism to store data for decades but was still extremely slow compared to today's standards. Advances such as core memory and magnetic disks paved the way for our modern data storage devices.

A modern computer uses two methods to store data. Data storage that is located extremely close to the processor and used for random access by the processor is called memory. Memory is typically volatile, meaning the data is stored only when power is applied to the device, and therefore is not a good choice for persistence. The non-volatile storage that is located further from the processor is called secondary storage which persist when power is removed. Our experiment used data from secondary storage devices. The two most prevalent types of secondary storage used today are magnetic hard drives and solid state drives.

### 2.4.1 Magnetic Hard Drives

Magnetic hard drives get their name because they are literally a rigid disk, called a platter, that is coated with magnetic material. The platter spins at over 7000 RPM and an actuator arm moves a magnetic head over the platter, allowing data to be stored on the surface of the platter [8]. The head magnetizes a microscopic section of the platter to record data. Since each square centimeter on the platter holds 31 billion bits, it is important to have an organized method to know where the data is stored on the platter [8].

During manufacturing, each platter is made with thousands of tiny concentric tracks on the surface and each track is divided into equal sectors [8]. This system allows data blocks that are recorded to the disk surface to be assigned an identification number called a Logical Block Address (LBA) which corresponds to a head number, cylinder number, and sector number [8]. This numbering system is used to locate the data stored on the disk. Although the amount of data that can be stored on the same size platter has increased with advances in technology and changes have been made to facilitate faster data transfer, the basic function of the magnetic hard drive remains the same.

### 2.4.2 Solid State Drives

On the surface, solid state drives (SSDs) are effectively integrated circuits and memory chips. The lack of moving parts increases the read / write speed. The complexity of SSDs lie in their operation, not the physical components. SSDs read and write data to a non-volatile memory called NAND flash memory [9]. One reason for the drastically increased read / write speed is the drive's built in controller which manages addressing and computing shortcuts for data throughput [9]. There is another key difference about the operation of the solid state drives: their read and write operations work on many bytes at once. Data is arranged into blocks and each block is divided into pages. Hutchinson [9] explains that because of its design, an entire block must be written or erased at the same time.

The TRIM command allows the file system to communicate with the controller to clean up unused blocks, for the purpose of optimizing write speed. Hutchinson explains that this is required because the NAND flash memory cannot be overwritten during the write process [9]. The page must be empty (all 1s) in order to allow data to be written to the page [9]. The function of the TRIM command is to automatically delete (make all 1s) the contents in a page when the user or operating system deletes the corresponding file. The downside to SSDs is that they can support only a limited number of writes to each individual page cell before it can no longer physically store a zero charge and therefore becomes useless. Implementation of the TRIM command helps to reduce the number of unnecessary read / writes and therefore prolong useful life of the SSD.

## 2.5 Forensic Tools for Analysis

We used forensic analysis tools and techniques to image the secondary storage devices, and produce the raw forensic image we needed to conduct our experiment. The technique we used is outlined in Section 4. Each tool plays a role in overcoming the variety of operating systems and configurations we encountered during collection of the participants' drives. In this section, we explain each tool's functionality and briefly give the reason for use in our research. Because we use a bootable USB drive with Ubuntu operating system to bypass the different operating systems and file systems on the participants' drives, we were able to use the same tools and techniques on every drive we encounter. The drive imaging tool we use is a command line version of Forensic Took Kit (FTK) Imager.

**FTK Imager**

FTK Imager is a digital forensics tool that can create a raw forensic image of a drive. FTK Imager stores an image in several formats, to include raw format (dd), EnCase format, or

SMART's file format [10]. The format that we chose for the participant drives was the *.E01* format from EnCase. The options for FTK Imager allows for output to a single file or output to several smaller files depending on which commands are given on the command line. The command line tool provides additional functionality for compression of the files, with "0" being using no compression to "9" using the most compression. We split the images into 15 gigabyte files during the imaging process.

**bulk_extractor**

Once we acquire the forensic image using FTK Imager, we used a open source tool called bulk_extractor. The computer forensic tool, bulk_extractor, "operates on disk images, files or a directory of files, and extracts useful information without parsing the file system or file system structures" [11]. bulk_extractor is capable of parsing several formats such as PDFs, RARs, URLs and JPEGs. It is capable of running multi-threaded and has many options for customizing its behavior. The output that we used in our experiment resides in the feature file named "email.txt", which contains the offset of the email addresses on the drive, the email addresses, and a "context" field showing the contents of the bytes on either side of each address.

**friend.py**

After we extracted the email feature file from the images we ran the algorithm friend.py. The code for friend.py is listed in Appendix A. The algorithm friend.py produces an output that is in GEXF (Graph Exchange XML Format). GEXF is "a language for describing complex networks structures, their associated data and dynamics" [12]. An additional format we used with the output of friend.py is the Graphml format, which is another format based on XML.

The algorithm friend.py takes as input a path to a directory or a text file with a list of multiple directories containing the output from bulk_extractor. It also requires a path to a directory on the command line to output the results from friend.py. Some of the arguments that can be processed on the command line by friend.py are:

- Window size - the default is 128-bytes but size can be 256-bytes

- Big-subgraphs - subgraphs are separated by the largest connected components and this selects how many components will be written to separate files. The default is ten.
- Fixed Window - The default is set to false to produce a sliding window based on the window size

The algorithm, friend.py, saves the nodes and edges to dictionaries. Using the window size of 128-bytes, friend.py will store any two nodes within the sliding window into a pair, making an edge between the nodes. The window will stay open and continue to link the nodes until there is a window-length space without an email address, closing the window. As friend.py moves through the bulk_extractor output and finds another email address a new window will start and link the nodes within that window. An additional function in friend.py is to apply a weight to pairs of nodes that are seen more than once. For example in Figure 2.2 node $a@mail.com$ and node $c@mail.com$ have two different occurrences, so a weight of two is applied. In Figure 2.3, the edge between node $a$ and node $c$ is thicker indicating multiple occurrences.

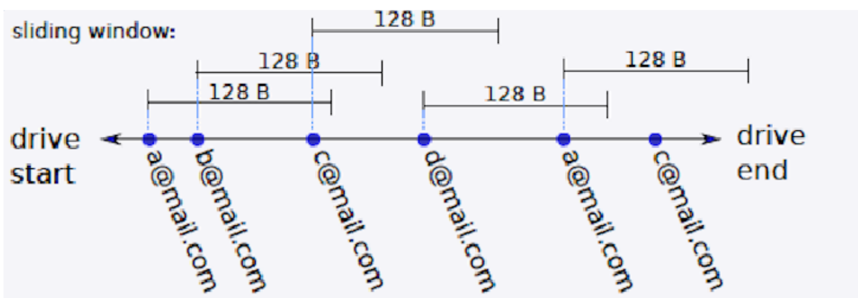


Figure 2.2. friend.py showing a 128-byte window. Source: [13].

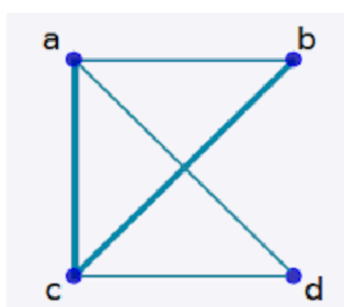


Figure 2.3. Graph resulting from 2.2 based on a 128-byte window. Source: [13].

**identify_filenames.py**

A tool that comes with bulk_extractor is a Python script called identify_filename.py that produces the path or context around an email address. Using identify_filename.py with the email.txt feature file from bulk_extractor and the image of the drive adds annotations to the email.txt file which provides the location in the file system and the md5 hash of the file where each email address was located. The tool identify_filename.py was used to understand the location of "not useful" email addresses on the drive (see Section 5.3.2).

## 2.6 Network Science and Graph Theory

Network Science is a field that has emerged within the twenty-first century. As described by Barabasi [14], Network Science looks at complex networks, such as, cellular networks, neural networks, social networks, or power grids. Analysis of complex networks, such as social networks, uses the characteristics of the networks and graph theory to better understand the networks. In our experiment we used several techniques from graph theory to analyze, visualize, and compare graphs.

The start of graph theory dates back to 1736, proving its usefulness in solving questions involving bridge crossings [15]. The first problems in graph theory showed the importance and usefulness of modeling data by using a graph and visualization. In this section, we explain some key concepts we used to evaluate our graphs and how graph theory concepts help us to identify useful information in our experiment.

### 2.6.1 Graph Attributes

The definition of a graph is the following: "A *Graph G* is an ordered pair of finite disjoint sets $(N, E)$ such that $E$ is a subset of the set $N \times N$ of unordered pairs of $N$. The set $N$ is the set of *[nodes]* and $E$ is the set of *edges*. If $G$ is a graph, then $N = N(G)$ is the node set of $G$, and $E = E(G)$ is the edge set. An edge {x, y} is said to *join* the nodes $x$ and $y$ and is denoted by $xy$. Thus $xy$ and $yx$ means exactly the same edge; the nodes $x$ and $y$ are the *end nodes* of this edge" [16]. We will use the term node to refer to each email address that exists on the graph. A node can also be referred to as a vertex.

In Figure 2.4, the nodes are labeled $n1$ through $n9$ and the edges are labeled $e1$ through $e9$. A connected graphs means that all of the nodes have edges that are adjacent to each other, such that $G$ contains a $u - v$ path for every pair of $u, v$ of vertices of $G$ [17]. Another important characterization we use is the concept of components in a graph. Components assist in making graphs more manageable to analyze by breaking the graph into sections of connected graphs. The graph $G$ may have several components that are not connected to each other. In Figure 2.4, we show a graph $G$ with three components or subgraphs, $H$, $J$, and $K$. In our experiment, these components or subgraphs are individually analyzed to assist with graphs with high node and edge counts.

An edge that occurs between nodes $x$ and $y$ is counted as a weight of one. If the edge occurs multiple times between nodes $x$ and $y$, the weight increases incrementally by the occurrence of the edge. In the Figure 2.4, edge $e6$ has a weight of one since the edge occurs one time. If the edge between the nodes $n5$ and $n6$ occur again, the weight would be two for the edge $e6$. Weight is visually represented by a thicker line compared to edges with smaller weights. An example of weight is shown in Figure 2.3 between node $a$ and $c$.

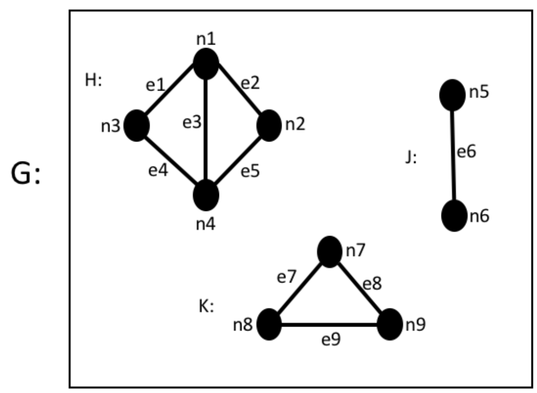


Figure 2.4. Graph $G$ and its components named H, J, and K

### 2.6.2 Isomorphic Graphs

Isomorphic graphs are graphs that have the same structure, meaning they are the same graphs but look different visually [17]. In Figure 2.5, we show two graphs that look different but would be identical if the nodes were in a different layout. These graphs are small examples in comparison to the graphs in this experiment. However, understanding if a graph is isomorphic is useful for categorization of the graph.

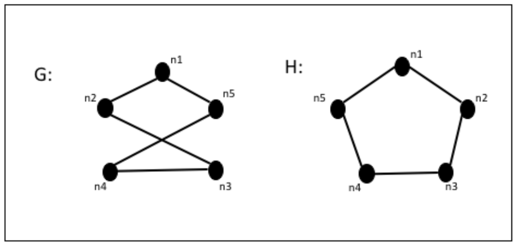

Figure 2.5. Isomorphic Graph Examples. Adapted from: [17].

### 2.6.3 Node Characterizations

We take into account two characteristics initially when analyzing each subgraph: size and order. The number of nodes of a graph is referred to as the order of $G$ and the number of edges is referred to as the size [17]. In Figure 2.4, graph $J$ has an order of 2 and a size of 1. Order and size are important to understanding other concepts such as degree of the nodes or weight of the edges in a graph. We also use a number of attributes that are specific to each node in the graph. These include degree, eigenvector centrality, modularity and $k$-core.

**Degree**

The degree of a node is the number of edges that are incident to the node, which is annotated as $deg_G v$ [17]. In Figure 2.4, the degree of node $n7$ in graph $K$ is 2, while the degree of $n1$ is 3. We find that supplementing the information that the degree captures with the eigenvector centrality of a node is useful in identifying important nodes in the network.

**Eigenvector Centrality**

While the degree of a node captures the number of neighbors a node has, eigenvector centrality of a node is the value the node receives based on its neighbor's importance or influence in the network. We use this measurement to decide who the drive owner could be most closely associated with in a subgraph. To understand eigenvector centrality, consider an adjacency matrix to represent the graph. An adjacency matrix is a $nxn$ symmetric matrix, where $n$ is the number of nodes in the network [18]. If there is an edge between two nodes $i$ and $j$, then the element $a_{i,j}$ in the adjacency matrix is 1 otherwise it is 0, shown as:

$$A_{ij} = \begin{cases} 1, & \text{if there is an edge between nodes } i \text{ and } j \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Nodes with high degrees in a social network commonly have more influence in that local neighborhood of the network. Using degree, weight of the connections between the nodes and the understanding that the connections are not equal we can arrive at eigenvector centrality [18]. The centrality of a node $i$, denoted as $x_i$, will allow for "making $x_i$ proportional to the average of the centralities of $i$'s network neighbors

$$x_i = \frac{1}{\lambda} \sum_{j=1}^{n} A_{ij} x_j, \tag{2.2}$$

where $\lambda$ is a constant [18]".

**Modularity**

Modularity in a graph measures the quality of a partition of a graph [19]. A high modularity in a graph means that fragmentation into communities is possible, where a community refers to a collection of nodes in which there are dense connections between the nodes in the same community, and fewer connections between the communities. The algorithm we use implemented in Gephi is described in Section 2.7. It was first proposed in "Fast unfolding of communities in large networks" by Blondel et al. (see Section 3.4.2). Blondel et al. describes the formula for modularity as:

$$Q = \frac{1}{2m} \sum_{ij} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j), \tag{2.3}$$

"where $A_{ij}$ represents the weight of the edge between $i$ and $j$, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex $i$, $c_i$ is the community to which vertex $i$ is assigned, the $\delta$-function $\delta(u, v)$ is 1 if $u = v$, and 0, and $m = \frac{1}{2} \sum_{ij} A_{ij}$" [19]. The modularity is is a value between $-1$ and 1. The algorithm repeats two steps iterating through the nodes until all the nodes belong to a community where the maximum modularity for that community is reached. Finding communities within a graph helps us visualize the clusters within the graphs.

**k-core**

The $k$-core of a network is the part of the network in which every node has a minimum degree of $k$. For example, if a node represents a person and an edge represents friendship, nodes in the $k$-core have at least $k$ other friends in that part of the network [20]. $k$-core is useful in reducing noise in the graph and making the useful email addresses visible.

## 2.7 Visualization Tool and Analysis

Gephi is an open-source tool we used to visualize subgraphs from participants' drives during the experiments. Gephi is visualization and exploration software for graphs and networks that allows for manipulation of the data to identify structures and patterns [21]. Algorithms built into Gephi allowed us to use for multiple metrics and layouts to best visualize our data. In particular, we used $k$-core to filter the graph (see Section 2.6.3) and modularity to partition the subgraph (see Section 2.6.3). Gephi is also capable of importing plugins to facilitate specialized manipulation and filtering of the dataset [22]. The user interface is shown in Figure 2.6.

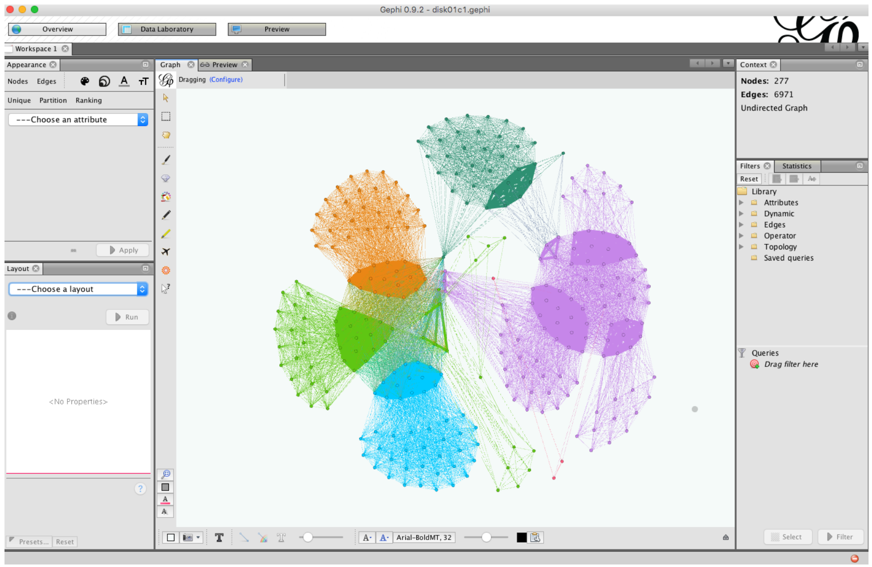


Figure 2.6. A graph in Gephi showing multiple modularity classes distinguished by different colors. The appearance and layout functions are found on the left of the graph. The statistics and filtering functions are found on the right of the graph.

## 2.8 Measure of Performance

The primary methods we use to measure performance are precision, recall, F-score and the confusion matrices. To derive these metrics, we first calculate counts of true positives, false positives, false negatives and true negatives. These counts provide initial insight into the effectiveness of our method. In our experiment, having high false positives would mean the methodology used is over classifying subgraphs as "useful." If we have high false negatives, we are missing subgraphs that are "useful." The counts provide the ability to build a confusion matrix and calculate precision, recall and F-score.

### 2.8.1 Confusion Matrix

The confusion matrix, or classification matrix, is built using the counts of true positives, false positives, false negatives and true negatives. In our experiment, we use a two-class confusion matrix, shown in Table 2.1.

Table 2.1. Example of Confusion Matrix

| | | Predicted | |
|---|---|---|---|
| Actual | | Useful | Not Useful |
| | Useful | True Positive | False Negative |
| | Not Useful | False Positive | True Negative |

### 2.8.2 Recall, Precision and F-Score

Recall is defined as the proportion of positive cases that are predicted positive. It is also referred to as sensitivity or the true positive rate. The higher the false negatives, the lower the recall score and the lower the false negatives the better the recall will be for the method being measured. The formula for recall is as follows:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}. \tag{2.4}$$

Precision is the proportion of predicted positive cases that are actually positives. It is also referred to as the positive predicted value [23]. The formula for precision is as follows:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}. \tag{2.5}$$

The F-score represents the harmonic mean of the precision and recall. The formula for F-score is as follows:

$$FScore = 2 \times \frac{precision \times recall}{precision + recall}. \tag{2.6}$$

# CHAPTER 3: Related Work

In this chapter, we review work discussing the methods and tools used in the analysis of bulk data, email addresses, and social networks, as well as graph classification techniques. Our research builds on and continues the work of two Naval Postgraduate School students.

## 3.1 The Forensic Process and the Volume Problem

The digital forensics process involves more than just finding the data. There are many tools available to accomplish this task. Since the output from these tools will commonly be used as evidence to present in court, it often must meet certain legal requirements for admissibility [24]. There are numerous process models used to conduct digital forensic investigations [25]. The digital forensic process has evolved as technology has advanced. The Abstract Digital Forensics Model [26], Enhanced Digital Investigation Process Model [27], Forensics Process Model [28], Investigative Process Model [29] represent a few iterations during the growth of the field.

Each of these models emphasizes different approaches. In 2002, Reith et al. [26] present an Abstract Digital Forensics Model. They define common steps used in other forensics models and produce an abstract model that can be used independent of a particular technology. The Enhanced Digital Investigation Process Model, presented by Baryamureeba and Tushabe [27] in 2004, uses five phases to redefine the progression of the forensic process. Their goal is to improve the methodology and approach to the investigations and increase criminal convictions on computer related crimes. In 2006, Petroni et al. [28] introduce the Forensics Process Model which provides an extensible framework that makes volatile memory forensic analysis more practical. The Investigative Process Model, introduced by Freiling and Schwittay in 2007, places emphasis on the scientific approach and the careful handling of digital evidence [29]. In 2012, Kohn et al. introduce the standardized Digital Forensic Process Model to assist in digital investigations [25]. Because our research is based on data gathered for investigative purposes vice the presentation of trial evidence, we will focus on the analysis phase as described in many of these models. The analysis phase includes identifying relationships between data fragments, determining the significance of the information, and arriving at proper conclusions [30].

Regardless of the model used to approach the digital forensic process, the common problem that must be addressed is the shear volume of data [31]. Andrzejewski [31] discusses the rapid growth in storage capacity and the problems it poses for digital forensic analysts. He proposes a method using open source software to distribute the analysis of the raw forensic images [31]. His work can assist with the volume problem and we believe automating steps in the forensic process will assist in processing the huge amount of data in a timely manner.

## 3.2 Forensic Data Analysis Tools

Digital forensics can provide investigators with accurate, relevant information in a timely manner. This, however, is not always an easy task. Two major obstacles are technological obsolescence and manpower constraints [32]. The rapid rate at which technology is advancing has turned some forensic tools into obsolete relics and Garfinkel [32] argues that the lack of formal standardization will make digital forensics substantially less effective. These two obstacles are connected based on the fact that new technological advances in digital forensics requires personnel to understand the technology. Furthermore, with the increase of the volume of data requires manpower to analyze the data. Individual file extraction and analysis is a common forensic practice due largely to the vast array of digital storage mediums, file systems, and formats in use [2]. This burdensome process requires a large amount of analyst time and can create a backlog of data awaiting analysis [1]. In some cases, a forensic analyst can mount a hard drive to their computer and use a standard operating system to view allocated files. This is the simplest method of digital forensics but not the desired approach due to possible alteration of evidence and the time it takes to conduct analysis. The obstacles presented in the digital forensic field could be reduced by using current tools to process the forensics data.

During the analysis process, forensics analysis use tools based on what needs to be investigated. Carrier [33] uses abstraction layer theory to analyze forensic tools and propose a set of requirements for forensic analysis tools. As discussed in the previous section, the various frameworks outline methods for memory forensics dealing with volatile and non-volatile forensic data. Our research focused on non-volatile data using a method independent of the operating system. Operating systems use different file systems which could complicate the forensic process based on the tools used [34]. In analyzing these files, specialized forensic software and additional tools are required if the operating system is unknown or if the analyst is looking for unallocated files [33]. The solution we implement to standardize our process across all operating systems was bulk data analysis.

Several bulk data tools exist to extract data for forensic analysis, such as Encase, Forensic Toolkit, and The Sleuth Kit. All three of these are capable of parsing the most commonly used file systems. Encase has been widely used by police departments around the US for decades [35]. It makes a bit stream mirror image of the drive and offers an integrated set of forensic tools to analyze the data [35]. Forensic Toolkit (FTK) provides law enforcement and corporations an integrated kit of tools capable of reading, acquisition, decryption, analysis and reporting of digital evidence [36]. We utilize FTK Imager to create raw images of the participant drives while leaving the underlying data undisturbed. The Sleuth Kit (TSK) is an open source software with 21 very powerful Linux based tools [36]. A benefit of TSK being open source is the peer review and scrutiny it has undergone, which allows it to inherit certain security principles [36].

### 3.2.1 Bulk Data Analysis

Bulk data analysis can be used as an initial step to determine if the media contains valuable intelligence data [2]. Bulk data extraction attempts to ignore the interfaces rather than parse them by working at the block device level. The block device is shown in Figure 2.1, showing each level from applications to storage. Using bulk data extraction techniques allows prioritization for deeper analysis and results in more significant information generated during a given amount of analyst time [2]. However, bulk data extraction produces higher levels of noise within the data. Bulk data analysis finds significant features of specific types of data that an analyst would be looking for, such as email addresses, credit card numbers, and search terms [2]. Garfinkel [2] introduces the bulk_extractor tool and shows how it is more effective in gathering high-value forensic features than current tools. Discerning the value of the information contained on a drive dramatically increases efficiency and allows an analyst to focus on the valuable data.

### 3.2.2 Sifting Through the Data

During a digital forensic investigation, there is a good chance that much of the data on a hard drive undergoing analysis is not useful [37]. Some data is just "uninteresting" if it provides no information about the owner [37]. The task of finding the desired data and discarding the uninteresting data can be manually tedious. If automatically identifying interesting data is difficult or impossible, getting rid of data that is known to be uninteresting allows for more efficient use of time spent analyzing data that has the possibility of producing valuable information [37]. Rowe's work compares numerous automated methods of identifying uninteresting files and showed successful results [37].

## 3.3 Email Analysis

Personal hard drives and other storage media can be a trove of useful data during an investigation. Rowe et al. [38] point out that people may be guarded about the information they share on social media sites but there is little concern about the information being stored during routine computer use. Key pieces of data, such as email addresses, can be easily retrieved by searching the drive and these can be discovered regardless of the operating system used [38]. Developing social networks based on email addresses is made possible due to email's availability and frequent use.

There is a variety of email forensic tools available to an analyst. Each one may be better suited for a specific task. Five commonly used open source tools are MailXaminer, Add4Mail, Digital Forensic Framework, eMailTrackerPro, and Paraben E-Mail Examiner [39]. Each of these tools have varying capabilities and uses metadata to examine the content of the email files that are located on a hard drive. Our approach instead focuses on the physical location of the email address stored on the drive and not the email content. This is a result of the bulk analysis approach we use to extract email addresses.

Bulk_extractor finds features such as email addresses on storage media without using file system structures. However, the raw output is a list of tuples containing the byte offset, email address, and context field. Further analysis is required to identify important addresses or social connections. There are several ways in which connections between specific email addresses can be made. Rowe et al. [38], discusses how the placement of the email address on the drive, matching words, email address structure, and numerous occurrences of different email addresses located in close proximity to one another give an examiner the ability to determine the strength of the connection between the email addresses [38]. We use visualization software to further analyze the bulk_extractor output.

### 3.3.1 Email Evidence

Various types of forensic evidence is left on a users drive during the process of sending an email [40]. Even if the email has been deleted from the mail service and the "trash" is emptied forensic evidence is often still present on the drive. This is due to just the pointer to the data being deleted with no change to the data in storage. Although email transmission follows standard protocols, local email storage depends largely on the email client [40]. Boucher and Kuange [40] discuss two ways a user accesses email. First is through an email client system which operates using software on the user's computer. The second is through a browser-based email system which operates through a connection with an email server. Forensic evidence is left behind from both systems [40].

It is common for an individual to have more than one email address. This will present a challenge for the construction of the user's social network unless these email aliases are resolved to the actual user. Bird et al., ran into this issue during their analysis of email coordination and communication between open source software developers [41]. They used header data and an automated clustering process to group similar email addresses together and then manually processed those results [41]. Their clustering algorithm used five similarity measures to calculate an overall similarity score. If the score exceeded an empirically set threshold, the aliases would be clustered [41]. However, their method for determining aliases of the same user is only effective if the aliases contain similar features. We address this issue by requesting alias resolution information directly from the device owner during our interview process.

### 3.3.2 Visualization

Visualization tools help address the volume problem by providing analysts with a representation of the data that highlights important artifacts and accelerates the examination process. After removing the uninteresting addresses, there are different ways to visualize email networks. Measuring the dissimilarity between the email addresses proved to be key in visualizing connections [38]. Rowe et al. [38] use dissimilarities to correlate distance and then look for email addresses that clustered together. Different factors used by Rowe et al. [38] to explore dissimilarity are as follows:

- Dissimilarity of the words in the email addresses,
- Magnitude of the difference in offsets,
- Email addresses located in different files,
- Email addresses located in different messages.

Rowe et al. [38] also use similarities between different drives to draw on tens of thousands of data points and provide additional reliability. This generated a graphical representation of a social network based on the email addresses. This paper showed that email addresses alone, without the contents of the email, could be used to build a reliable social network for the user of the drive [38]. Green [3], discussed in Section 3.6.1, used byte-offset between digital artifacts on the storage media to establish links that represented communications between users.

## 3.4 Social Networks Analysis

A frequent goal of forensic analysts is to understand the social network of the device owner. This requires the ability to piece together digital artifacts to construct a model of the owner's social interactions. Social network analysis has been used since the 1930's when Jacob Levy Moreno developed sociometry [42]. He merged the social network analysis of the time with mathematical models and graphic imagery [43]. Contemporary social network analysis requires the ability to reconstruct complex relationships because users can represent themselves with different email aliases and through numerous social media channels. Fortunately, network science and graph theory offer many useful techniques for inferring and working with social network data. In particular, our work relies on degree of a node, Eigenvector centrality (defined in2.6), and community detection algorithms. We also make use of several different layout algorithms to present the data.

### 3.4.1 Inferring Social Networks from Email Data

Multiple studies have demonstrated the feasibility of constructing meaningful social networks from email corpora. Perhaps the best known examples use the Enron corpus, a collection of nearly half a million emails from 158 employees over a period of 3.5 years. [44]

In their 2005 paper on the Enron Email Corpus, Diesner, Frantz, and Carley analyze the patterns of communication by the Enron employees during the company's collapse [44]. They sorted and classified the emails to build the communication structure between employees based on their rank and positions. Diesner Frantz, and Carley attempted to understand the structure of the communication networks in Enron as the crisis unfolded and they examined patterns of senders versus receivers as well as comparing the email to the organizational structure [44].

The authors extracted several attributes from a database containing the corpus to graph the communication structure using DyNetML, an XML-based markup designed for social network data [44]. They placed edges to represent email correspondence between Enron employees and weighted the edges based on frequency. They analyzed the density, betweenness centrality, and degree centrality of the resulting graph to identify important agents in the network. They concluded that patterns and volume of communication changed during the crisis, leading to a break in traditional and formal emails between employees of different rank within the company.

Chapanond et al. used graph theory and spectral analysis to compare the Enron email data and Rensselaer Polytechnic Institute (RPI) email data [45]. They create communities and model the communication structure. The authors only use the "To" and "From" fields of the Enron data set, which covers 19 months and contains 150 employees email logs. The RPI email data set has 1681 vertices compared to Enron with 150 vertices. The metrics used were degree distribution, diameter, average distance, average distance ratio, compactness, clustering coefficient, betweenness, relative interconnectivity, and relative closeness [45].

The results suggest the two organizations have different structures. The Enron email graph follows the power law and the largest component of the graph contains 62% of the vertices. Visualization of the Enron graph shows differences in communication when related to pay structure within the organization. Although both the analysis and work by Diesner el al. successfully demonstrated the ability infer social networks from email data and allowed the authors to arrive at new insights about the corresponding social interactions, their process requires intensive manual analysis that is unlikely to scale to large investigations. In addition, they rely heavily on email header data to build their model; attempts to automate would therefore require recovering and parsing these headers.

### 3.4.2 Communities in Large Networks

One way to break down larger networks to analyze them, is to consider the network at the community structure level, where communities are groups of nodes that are more interconnected than interconnected to the other communities. Blondel et al. [19] developed the Louvain method that uses modularity optimization to construct large networks (see Section 2.6.3). The algorithm used is executed in two phases. First, each node is assigned a community. Then the algorithm iterates over all nodes. For each, it considers every neighboring community and decides whether the community would be stronger with that node. If the modularity increases, the new node joins the community. If the modularity does not increase, the node remains in separate a community. This process is repeated until there are no longer any modifications to the exiting partition of nodes into communities. The second phase of the algorithm builds the networks out of the communities identified in the first phase and repeats the process but now the modularity depends on communities merging (rather than a node merging into a community).

The advantages of the algorithm include unsupervised operation which is fast and easy to implement. The disadvantage is that it is not a deterministic algorithm, so the community detection generally produces different communities. During testing, the algorithm was applied to large networks. The speed of computation and modularity were compared to the results from three other algorithms. This validated algorithm is implemented in Gephi to assist visualization of the graphs nodes in our research.

## 3.5 Classification Techniques

The automation of secondary storage analysis relies on an accurate classification of graphs. That is, given a graph, analytical tools must be able to determine whether or not it is likely to be relevant to a forensic investigation. Machine-learning approaches to this problem typically rely on the ability to represent the objects being classified as a vector of features. Fortunately, graph theory offers a wide variety of potential features that can be calculated from the graph. These features can be used to enhance visualizations of a social network or help in classification of graphs with machine learning techniques.

### 3.5.1 Graph Fingerprints

Noting the difficulty of comparing unlabeled graphs with millions of nodes and edges, Bonner et al. [46] outlines a feature extraction approach called Graph FingerPrint Comparison (GFPC). This method compares large graphs that are topologically similar by extracting local and global level features [46]. To compare graphs, the GFPC approach begins by building a "fingerprint" from the local and global features extracted from the graph. To improve efficiency, node-level features are summarized using common statistical measures. Once the fingerprints are created, the authors perform similarity calculation using Canberra distance. However, since the fingerprints are essentially equivalent to feature vectors, they may also make a viable input for a machine-learning classifier.

### 3.5.2 Support Vector Machine Use on Email Data

Standard machine learning tools, such as support vector machines, have the potential to contribute to the automation of a variety of challenging forensic analysis tasks. Liu and Lee [47] demonstrate the effectiveness of SVM classifiers to conduct sentiment analysis the Enron Email Corpus. Their framework compares three different labeling methods, which include SentiWordNet labeling, k-means labeling, and Polarity labeling. They also test five different classifiers: SVM, Naïve Bayes, Logistic Regression, Decision Tree, and OneR [47]. After analysis of the labeling and classifiers, they propose using k-means clustering and SVM classification.

Labeling was required to provide the classification algorithms a set of validated ground-truth data to train against. Liu and Lee chose use k-means clustering, an unsupervised learning approach, to accelerate the labeling process. After clustering the emails, they apply sentiment labels to the clusters. These are then used to train their SVM classifier [47]. They tested their framework on the Enron email corpus.

Using a confusion matrix, they classified the labels as positive, negative, or neutral. They also measured the performance of the SVM classifier with respect to precision, recall, f-score, and accuracy (see Section 2.8.2). They found that SVM had the highest performance curve which indicated it was the best method for classification. We note that the success of the authors' approach depends on the availability of a relevant dataset of significant size. The absence of a similar dataset with contemporary machines for digital forensics contributed to this motivation of this research. Moreover, while the authors' goal of sentiment analysis requires only 3 types of labels, classification of email networks is considerably more complex. Both a robust taxonomy and reliable ground-truth labels are needed for the future development of machine-learning classifiers.

## 3.6 Constructing Social Networks from Secondary Storage

Two prior Naval Postgraduate students, Janina Green and Gregory Allen, made advancements toward reducing noise and extracting the device user's social networks. The algorithm they test defines a fixed byte window and constructed graphs of email addresses with links between addresses that fall within that window.

### 3.6.1 Constructing social networks from secondary storage with bulk analysis tools

Green [3] develops a method based on the byte-offset between digital artifacts on the storage media. The byte-offset approach was shown to establish links between users that represented communications [3]. Her analysis indicates that using measures of centrality assists in identifying important nodes and close associates. This information was used to create useful graphs which assisted in establishing the social network of the owner.

Green [3] uses three groups of drives for her analysis. She has two data corpora and ten collected drives from volunteers. Green's [3] approach is to analyze the drives, create graphs, and separate them based on usefulness. She conducts interviews and gets feedback on her results. Green [3] successfully uses automated steps to produce information valuable for an analyst building the user's social network.

### 3.6.2 Constructing and classifying email networks from raw forensic images

Allen [4] conducts several experiments using the same drives in Green's work. He classifies graphs of email addresses into the categories "useful" and "not useful". He uses Naïve Bayes, classification tree, logical regression, and support vector machines (SVM) to measure the data.

In the first experiment Allen uses all of the graphs and attributes in his data set to classify the graphs “useful” and “not useful”. In the second experiment, evaluated the effect of altering the maximum distance (in bytes) between email addresses that are linked in the network that the algorithm constructs. His third experiment used a selected set of attributes of the graph.

The attributes he finds useful are “density, average neighbor degree, Pearson coefficient, transitivity, highest betweenness centrality, maximal matching (divided by number of edges), maximal matching, number of nodes, degree distribution, and degree distribution value“ [4]. In his last experiment, Allen categorizes graphs into classes based on the type of email address.

Allen concludes that using the graph’s topological structure, he could classify the graph’s usefulness. The second experiment shows that using a 128-byte window performed better in correctly classifying the graphs. He identifies the top attributes to accurately classify the graphs. Allen identifies that using a labeling scheme better categorized networks to assist machine learning algorithms [4].

In our experiments, we build on the previous work of Green and Allen by identifying user communication patterns using a new algorithm and labeling a data set that could be used to train a classifier.

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 4:
# Methodology

Our experiment centered around analyzing the proximity of email addresses on secondary storage devices using the algorithm, friend.py. To perform this analysis, we first built a dataset of drive images collected from a pool of volunteers consisting of Naval Postgraduate School students, faculty, and friends. Then, using some of the successful techniques from previous Naval Postgraduate School thesis students, Green and Allen (see Section 3.6), we created a new methodology to determine the social network of the drive owners.

## 4.1 Identifying User Communication Patterns

In this section, we give an overview of the tools and methods used in our process, from collecting the raw forensic images and extracting email address to classification of graphs and evaluation of our analysis. In addition, we discuss the new algorithm friend.py and the process we use to visualize the subgraphs created in this experiment. Figure 4.1 is a visual representation of methodology. The steps in our methodology are:

1. Collect secondary storage devices from participants and create a raw forensic image.
2. Record general metadata about each drive and the email activity of the participant.
3. Process images using two tools: bulk_extractor and friend.py.
4. Use Gephi to run statistics and layout on each subgraph from the participant drives.
5. Classify the subgraphs as “useful” or “not useful” with respect to information the graph contributes regarding the participant’s social network.
6. Conduct interviews with the participants to find ground truth.
7. Inspect subgraphs or drives that require further analysis.
8. Measure effectiveness of our approach by comparing our labeling to ground truth interviews.

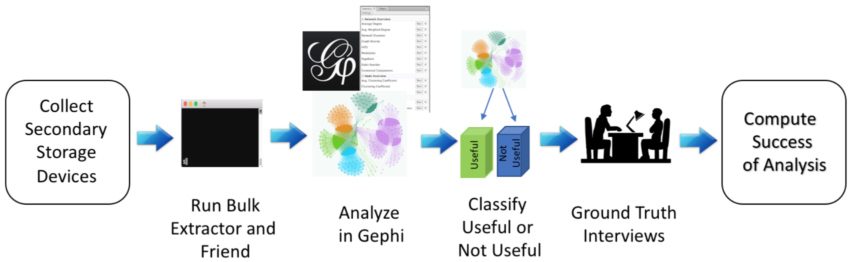


Figure 4.1. Methodology for the Experiment: Identifying User Communication Patterns. Gephi icon sourced from: [48]

## 4.2 Drive Collection and Email Address Extraction

To acquire drive images of our participants' machines, we used a bootable USB drive running the Ubuntu operating system. By operating from this external drive we were able to bypass the different operating systems and file structures on the participants' computers. Using the same operating system for imaging each drive allowed us to use a standard procedure each time and not interact with the file system or operating system on participant drives. We performed image acquisition using FTK Imager (see Section 2.5), which creates a raw forensic image and copies it to an external drive. An example of the FTK Imager command we use on a drive is:

```
ftkimager <src device> "<dst device>" -e01 -frag 15G -compress 9
```

The command takes the source device and copies the image to the destination device using *.E01* format. The image is fragmented into 15 GiB sections and compressed with the highest compression FTK Imager offers. Once the image is created, we run bulk_extractor to extract the email addresses and the offsets on the image. The command we used in bulk_extractor is:

```
bulk_extractor -o <output dst> -E email -j 4 <file_name>.E01
```

We executed the command against all *.E01* files created by FTK Imager. This bulk_extractor command produces several reports. The output that was relevant to our experiment is the text file *email.txt*. *Email.txt* contains all email address found on the drive, as well as the byte offset where they were found on the secondary storage device, and a context field showing the content of the bytes on either side of the address. The next step was to run the new algorithm, friend.py (see Section 2.5 for description or see Appendix A for Python code). The output from friend.py contains 11 GEXF files to be analyzed in Gephi, and a summary text file. Of these 11 files, one contains the entire graph produced by the friend.py algorithm, and the other 10 consist of the 10 largest connected components (by node count), each saved to a separate GEXF file. We use the top 10 components or what we refer to as subgraphs for our analysis. The command we used for friend.py is:

```
friend.py -p <bulk_extractor input> -o <output_directory>/
```

## 4.3 Analysis of Subgraphs

In preparation for analysis and participant interviews, we used Gephi to conduct the following steps on each of the 450 subgraphs:

1. Calculate the following statistics on the subgraph: average degree, modularity, and eigenvector centrality
2. Color the nodes and edges by community
3. Run the Fruchterman Reingold layout function (only if the total node count is below approximately $2,000$ nodes on the subgraph)
4. Locate the node with the highest degree
5. Locate the node with an eigenvector centrality of 1.0 to identify the most important email address in the subgraph
6. Classify the graph as "useful" or "not useful" based on visual inspection and statistics.

### 4.3.1 Setting up Graphs in Gephi

The Fruchterman Reingold layout function assists in visualization of graphs in Gephi. We apply this algorithm prior to labeling each subgraph "useful" or "not useful". The algorithm relocates the nodes and edges to meet a defined list of criteria. Which consists of evenly distributing vertices, minimizing the edge crossings, making the edges uniform in length, and conforming to the frame [49]. This algorithm has the benefit of making clusters within subgraphs easier to see, as well as clearly showing the nodes and vertices. Subgraphs over approximately $2,000$ nodes require additional steps prior to running the Fruchterman Reingold layout function due to computation time the algorithm requires. The additional steps for subgraphs with high node counts are described in Section 4.3.2.

After calculating average degree and eigenvector centrality, we used Gephi's “data lab” view to look for the node with the highest eigenvector centrality and the highest degree. This metric is a tool we used to identify if the subgraph was “useful” and to identify the drive owner.

During the participant interview, we used modularity classes and k-core filtering to highlight important features of graphs with high node counts.We use additional steps to filter graphs with high node counts using modularity classes and k-core during the participant interviews. Using modularity class filtering, we could discern if the communities accurately clustered social networks into groups that made sense to the participant. We explain this in more detail in the next sections. In Figure 4.2, we show a high node and edge count that requires filtering.

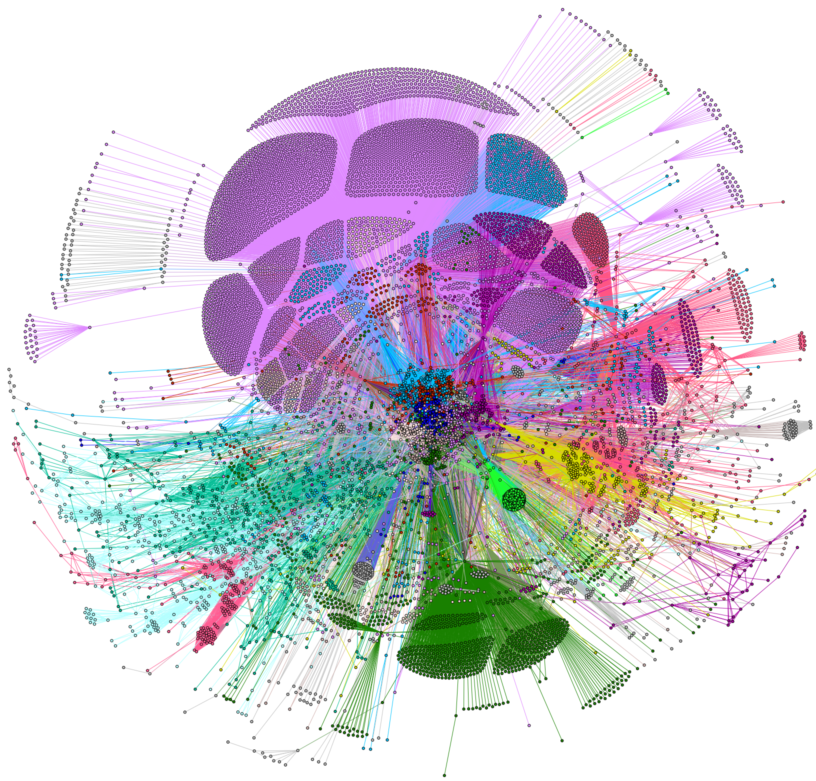

Figure 4.2. Disk11c1 has $10,937$ nodes with $355,567$ edges. Subgraphs of this size create computational issues running the Fruchterman Reingold layout in Gephi.

### 4.3.2 Filtering with Communities

We used the modularity classes or communities for filtering data in two different ways. First, we used the community filter to identify the "useful" subgraphs prior to the ground truth interviews. Second, during the interviews, we use modularity class to highlight the structure of subgraphs with high node counts to allow the participants to visualize the connections represented by the subgraph. If the participant identified a community as "useful," we would create a new workspace in Gephi for that community. This allowed us to further analyze it outside of the original subgraph. Calculating average degree, modularity, and eigenvector centrality on the community would further cluster the subgraph into new sub-communities and revealing more detailed structure within the original community. We applied new modularity class colors to the community and continued analysis with potentially fewer "not useful" email addresses in the participant's useful community. Figure 4.3, shows one community (purple) exported from the graph shown in Figure 4.2. After calculating the degree, modularity, and eigenvector centrality on the exported community, we show new sub-communities in Figure 4.4.

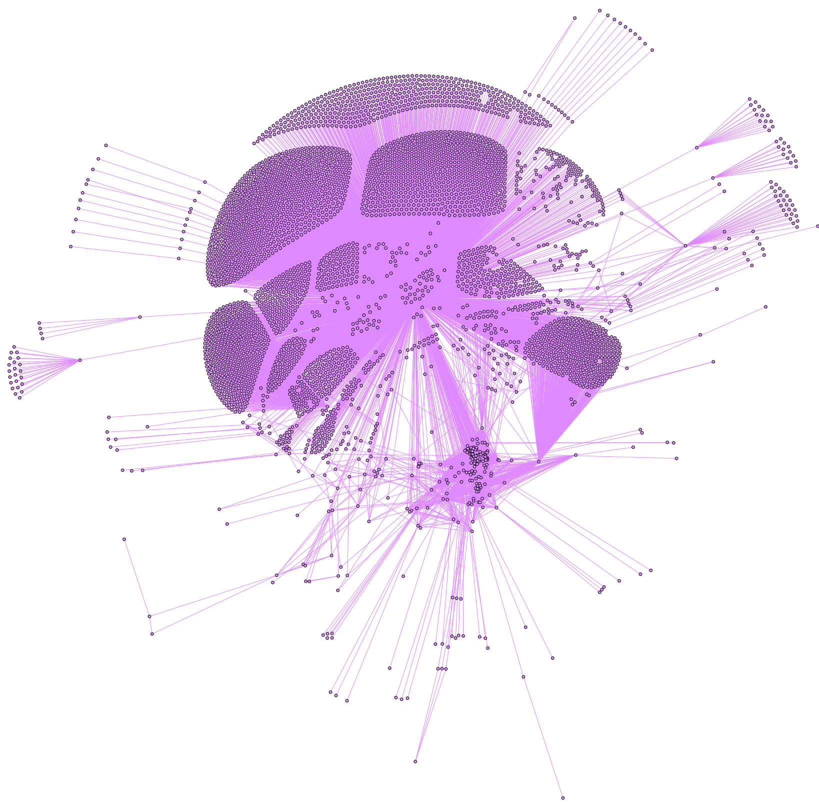

Figure 4.3. The community from the subgraph shown in Figure 4.2 that has the drive owner's primary email address is exported to a new work space. The export reduced the nodes to $4,265$ and the edges to $12,222$.

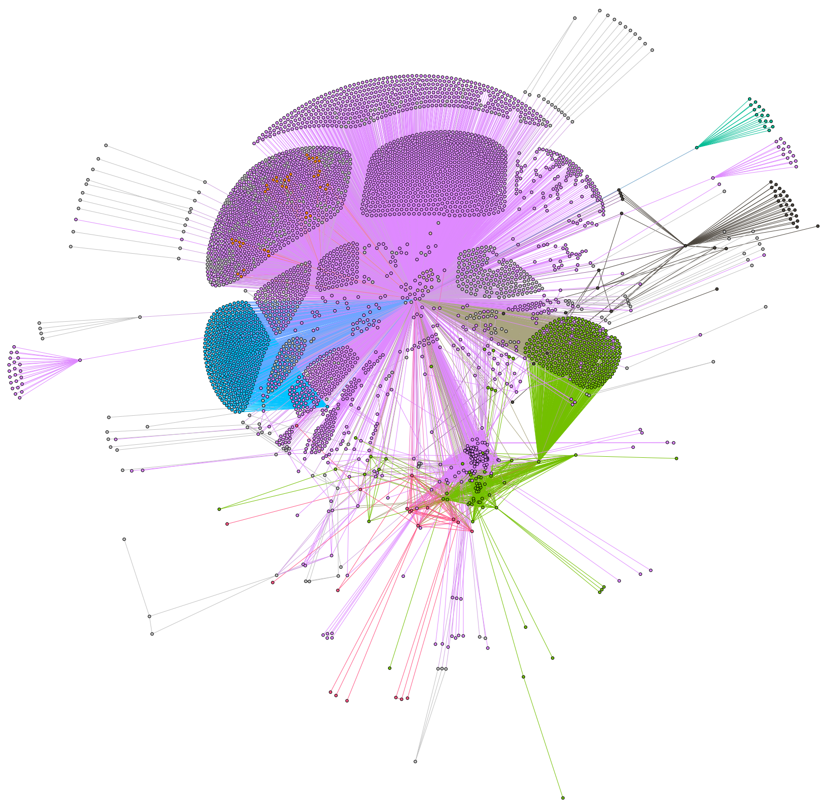

Figure 4.4. The subgraph shows what one community from disk11c1 looks like after running degree, modularity, and eigenvector centrality to refine the subgraph for analysis.

### 4.3.3 Filtering with k-core

After filtering by community, we used $k$-core (see Section 2.6.3) to remove nodes in the sub-community. This assisted in bringing out the social network in a subgraph with noise. An example of the type of nodes the $k$-core filter assisted in removing from the subgraph is message-id email addresses, such as `KAJSDOFU-ADFS-OWEKLNF-ASERJASLDF@GMAIL.COM`. These email addresses commonly had a degree of one and would be removed when the $k$-core filter was set to two. On large node count subgraphs, we use the k-core and modularity class filtering together to pull out the useful email addresses or clusters of email addresses. In Figure 4.4, we still show a large number of the message-id email addresses in Gephi. The $k$-core reduces the noise on Disk11c1 giving us the social network of the drive owner in Figure 4.5.

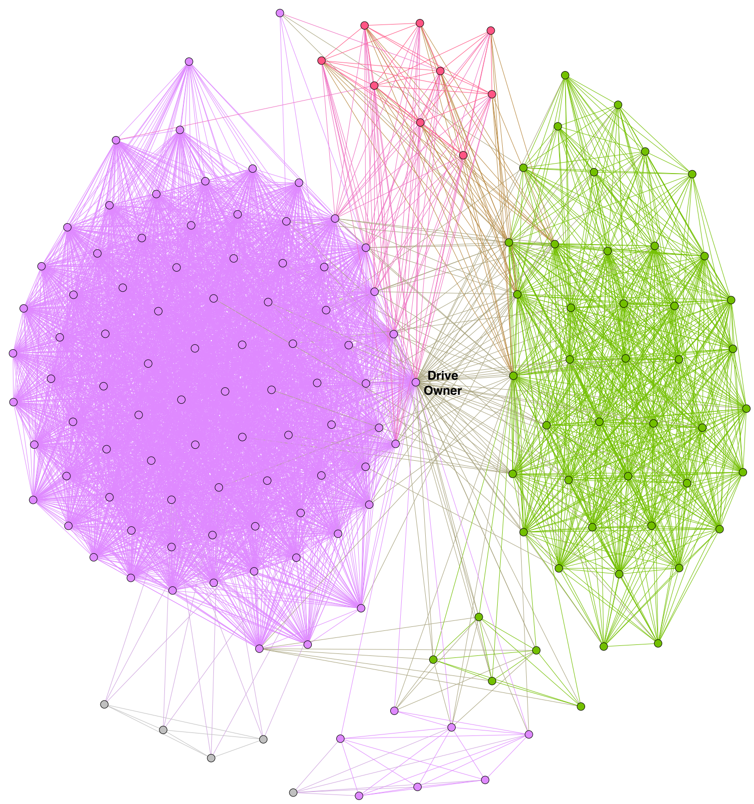


Figure 4.5. Disk11c1 subgraph after filtering with modularity class and $k$-core to reduce noise and visualize the social network. Each modularity class represents different social groups of the drive owner. The communities identified different time periods for the drive owner's social network.

## 4.4 Subgraph Labeling: "Useful" or "Not Useful"

We looked for common attributes on each subgraph to label the subgraph "useful" or "not useful". The following attributes assisted us in labeling each subgraph "useful" or "not useful" :

- Highest degree,
- Eigenvector centrality closest to 1.0,
- The same node with highest degree and eigenvector centrality,
- Type of domain of the email address,
- Names of common software found in the subgraph,
- Websites in email address across entire subgraph,
- Domain ending in ".pn",
- Completely connected subgraph containing common domain name.

We labeled almost all the subgraphs within our data set using just one attribute. However, labeling subgraphs with a high number of nodes required the use of multiple attributes simultaneously. Applying labels to each subgraph prior to attaining ground truth during the interviews was an important part of assessing the algorithm friend.py. We labeled a subgraph "useful" if it could be used to build the participant's social network, meaning it included someone they had emailed or who had emailed them. A "not useful" subgraph does not contain any email addresses that were used for communication by the participant. An example of a "not useful" subgraph is one that contains email addresses from the installation of software. If the subgraph was "not useful" we would continue to analyze the subgraph and apply a more detailed category label (see Section 4.4.2 below).

### 4.4.1 Inferring Labels from Attributes

The degree and the eigenvector centrality of each node provided a starting point for the analysis of each subgraph. If the same node has the highest degree and eigenvector centrality we concluded that node was the drive owner. We also evaluated the types of domains common within the subgraph. If we saw the same domain on each email address within a subgraph and it was not familiar, we conducted further research to identify the domain. An example would be if all of the domains end in a common software package. It would be labeled as "not useful" and we would annotate the name of the software. For any subgraph we labeled "not useful", we attempted to identify a common characteristic in the email addresses and recorded a more detailed category for that subgraph.

### 4.4.2 Subgraph Categories

Once the subgraph is labeled "not useful", we further analyzed it and assigned it to one of several categories. The category names are selected by a common domain or based on the directory path for the email addresses on the drive. A subgraph is given a name if it is seen more than once in the data set. The categories and associated description for the experiment are:

Table 4.1. Categories of Subgraphs in Experiment

| Subgraph Label | Description |
|---|---|
| Cert | A subgraph containing "@dot.ca.gov" for all nodes in the subgraph. |
| Contoso | A fully connected subgraph with the domain of "@contoso.com" for all the nodes. |
| Eldos | A subgraph containing "@eldos.com" for all nodes in the subgraph. |
| Documents | Completely connected subgraphs from email addresses found in Word or Excel documents. |
| Graphics | A subgraph containing ".pn" or ".jp" domain which are PNG files and JPEG files. |
| Libsamplerate | A subgraph containing an audio converter C shared library files with nodes ending in ".so" domain. |
| Libsndfile | A subgraph containing a reading and writing audio file C shared library with nodes ending in ".so" domain. |
| Microsoft | A subgraph containing Microsoft Office Contact list found in a file in the installation folder. |
| Social | A subgraph of the social network of participant. |
| Software | A subgraph of installed software including, Python, Debian, Apache, and other well-known software vendors. |
| Theora | A subgraph containing video compression and video CODEC software with the common word "theora" for the node labels. |
| Travel | A subgraph containing hotel, airlines, and other travel agency email addresses for node labels. |
| Unknown | A subgraph that does not have greater than one occurrence in the data set or does not fit into other categories. |
| VM | A subgraph containing node labels associated with installation of virtual machines. |
| Websites | A subgraph with node labels from cached websites or with a common domain for websites. |
| Whatsapp | A subgraph with node labels ending in "whatsapp" which is an instant messenger and VoIP software. |

## 4.5 Participant Interviews

The participant interviews consisted of asking 12 institutional review board (IRB) approved questions (listed in Appendix B). Each participant was presented with all ten subgraphs from their drive and asked to determine whether they were "useful" or "not useful", depending on whether they contained information pertaining to their social network. After identifying the "useful" subgraphs, the participants were asked if they recognized the email addresses of people with whom they use email to communicate and if their relationships with these people were accurately represented in the structure of the subgraph. That is, we asked if they could discerned if the clusters or pairs of email accurately reflected their social network. The expected response for these questions would be "yes" or "no" to simply identify, at a subgraph level, the usefulness of the subgraph.

Additionally, we asked the participants questions pertaining to their computer type, operating system, email provider, and email activity. Data points we wanted to track, such as if the hard drive was solid state or magnetic, are important to document for comparison when analyzing subgraphs. Another important data point we tracked was how the participants access their email. This documents key differences with respect to email access, such as the use of an email client installed on their computer, as opposed to browser-based email access. Participants were also asked to share if they were the only user of the computer or if they shared the computer, since this could affect identification of "useful" subgraphs. During the interview we noted any unusual subgraphs and allowed the participants to discuss what they thought they were. This led to identifying numerous subgraphs that required follow up analysis.

## 4.6 Additional Analysis on Subgraphs

To gain additional insight into subgraphs that required further analysis either during the initial set up or post-participant interviews, we used identify_filename.py (see Section 2.5). This tool provides the file or location that the email address was extracted from, which helped us to better understand subgraphs containing unusual or unexplained addresses. We used this method to find the associated program or file these address were extracted from and to explain why it was present in the data set. The command we used to run identify_filename.py is:

```
identify_filename.py -image <path to image> -featurefiles email.txt
             <path to feature file> <path for output>
```

This command adds annotations to the original email.txt file indicating the file name, if any, that is associated by the file system with the byte offset where the email address was found. After annotation was completed, we used the Unix command line utility grep to retrieve this information from the email.txt files. We then used computers with the same operating system to validate the results. For example, this procedure allowed us to find the file that contained the email addresses making up the "Microsoft" subgraph. Since Microsoft Office was installed on more than one computer we could verify that the the subgraph came from the same file in each case.

## 4.7 Measure of Performance for the Experiment

The measure of performance for the experiment is based on the quantity of correctly labeled subgraphs. The number of subgraphs that are "useful" that we correctly labeled "useful" prior to the ground truth interviews are counted as true positives. The number of subgraphs that are "not useful" that we correctly labeled "not useful" are counted as true negatives. Similarly, the subgraphs that are "useful" but labeled "not useful" are counted as false negatives, and the subgraphs that are "not useful" but labeled "useful" are counted as false positives. We report these counts in a confusion matrix (see Section 2.8.1). Additionally we calculated recall, precision, and F-score.

# CHAPTER 5:
# Analysis and Results

This chapter summarizes the analysis and results from our experiment. We start by describing the dataset used to conduct the experiment, and highlight trends in the metadata we collected describing the hardware and software used by the study participants. We then present metrics (precision, recall and F-Score) evaluating our ability to separate "useful" and "not useful" subgraphs using the analytical process described in Chapter 4. Subsequently, we give a detailed analysis of patterns in subgraphs that contribute to the taxonomy of graph categories we developed. Finally, we identify subgraphs that were unexpected or required additional analysis.

## 5.1 Aggregated Statistics on Participant Drives

We gathered information about the hardware and software components from each drive. The dataset consists of 51 secondary storage devices with 48 different participants. We were unable to use six drives due to errors that occurred during the acquisition process which reduced our dataset to 45 drives with 42 unique participants. We analyzed a total of 450 subgraphs from the 45 drives (10 from each drive).

### 5.1.1 Hardware Statistics

For each storage device we acquired, we recorded the age and manufacturer of the computer containing the device, the size of hard drive, and the type of hard drive (i.e. solid state or spinning disk). There are 27 magnetic drives and 18 solid state drives in our dataset. We initially hypothesized that the type of hard drive would be correlated with in the number of email addresses discovered on the drive, due to the fact that SSD on systems implementing the trim command aggressively clear out unallocated space. However, in practice we observed no such trend. The sizes of the drives ranged from 120 to 250 gigabytes for solid state drives, and 130 gigabytes to one terabyte for magnetic drives. The computers in our experiment were, on average, 2.73 years old. The newest computer was one month old and the oldest computer in our experiment was eight years old. The manufacturer of the computer was documented and is listed in Appendix C, along with all other recorded information on the drives.

### 5.1.2 Software Statistics

In the initial collection and during the interview process, we recorded the type of operating system installed on the computer, the type of email client, the method used to access email, how often the participant accessed email, and whether there were multiple unique users of the computer. The types of operating systems in our dataset are listed in Appendix C. We gathered this information prior to imaging the secondary storage devices but we also found that we could identify patterns in the subgraphs that would indicate whether the computer was running OSX or Windows. Further analysis on the impact of operating systems on social networks reconstruction may be worth investigating in future work. Many of the subgraphs on a computer with the OSX operating system had high node counts on the first few subgraphs and had several graphs containing the domain ending in “pn”. The “pn” subgraph will be discussed in more detail in Section 5.3.2. We considered high node counts for a subgraph to be over $8,000$ nodes.

The participants were asked if they used an installed email client or accessed email through a web browser. If an Outlook client was installed on a computer, the noise found in the “useful” subgraph is generally higher than noise found in the useful subgraphs of participants who use a web browser for email. The increased noise on the “useful” subgraphs requires additional filtering of the subgraph. If the participant used Apple mail, we could identify this activity by Apple’s use of email aliases. An example of Apple’s aliasing is <username>@mac.com, <username>@cloud.com or <username>@me.com. Each of these email addresses are separate in the subgraph but connected to the participant’s social network in different communities. The aliasing caused confusion when trying to identify the drive owners since the communications are split between the aliases. The participants commonly use both the client installed on their computer and web-based access methods. A complete list of email providers is listed in Appendix C.

Participants were asked how often they accessed email: daily, weekly, or monthly. The participants overwhelmingly check email daily. Out of 45 participants, 37 stated they check their email daily, five check email weekly, and three check email monthly. Finally, we had

11 participants that shared their computers with family or friends. This lead to partially unknown clusters of email addresses or subgraphs that some drive owners could not identify during the interview. However, in some of the cases, the drive owner could recognize the domain or email address of the other user and could classify it as the other user's social network. Overall, the the data collected from participants about their drives and email activity was useful in understanding how to process the data and in explaining similar subgraphs.

An additional question that we asked participants pertained to the use of VMs on their device. After analysis of the subgraphs we used identify_filename.py and found the email addresses were located in the VM files. We asked participants during the interviews if they used VMs and found that these subgraphs were found on devices of participants that used VMs. The use of virtual machines are recorded in Appendix C.

## 5.2 Effectiveness of our Method for Identifying User Communication Patterns

In this section, we compare the results of our process for labeling the subgraphs and the ground truth obtained during interviews to measure our ability to correctly label the subgraphs from the output of friend.py. Out of the 45 drives we analyzed during experiment, we found "useful" subgraphs on 44 drives. During the initial setup and analysis of the subgraphs, we correctly labeled 413 of the 450 subgraphs. We were able to successfully identify subgraphs that contained the participant's social network with a recall of 0.9677, precision of 0.6316, and F-score of 0.7643. The confusion matrix given in Table 5.1 shows a detailed breakdown of our predictions as compared with ground truth.

Table 5.1. Confusion Matrix showing the comparison of our subgraphs predicting labels to the ground truth categories determined during interviews.

| | Predicted | | | |
| --- | --- | --- | --- | --- |
| | | Useful | Not Useful | $\sum$ |
| Actual | Useful | 60 | 2 | 62 |
| | Not Useful | 35 | 353 | 388 |
| | $\sum$ | 95 | 355 | 450 |

Out of the 35 false positives shown in Table 5.1, all 35 were subgraphs that we concluded were user address books, which led us to categorize them as useful. However, during the ground truth interview the participants could not identify them as part of the social network. Therefore, these subgraphs are "not useful". A major factor contributing to this labeling error is that these subgraphs typically contained common domain names, such as, "Gmail.com" or "Yahoo.com."

The two subgraphs we labeled "not useful" that turned out to be "useful" were subgraphs we placed in the categories "VM" (Virtual Machine) and "Website." These two subgraphs are false negatives, meaning that in the ground truth interviews, the participant stated they were "useful" subgraphs. The reason our methodology mislabeled one of these subgraphs as "VM" during our initial analysis was its high node count, a common feature of subgraphs that were extracted from virtual machines. In addition, the subgraph contained many email addresses with common domains, and shared several other features typical of "VM" subgraphs (see Section 5.3.2). Similarly, the subgraph mislabeled as "Websites" contained several common domains that led us to label it as a "not useful" subgraph.

## 5.3 Characteristics of Common Subgraph Categories

In this section we describe "useful" subgraphs (categorized as social networks) and various categories of "not useful" subgraphs we identified on several different devices. The "not useful" subgraphs include software installations, C library shared objects, and PNG files. We also discuss the removal of noise from the social network subgraphs and the few instances when a "not useful" subgraph is embedded in a "useful" subgraph.

All of the participants but one could recognize email addresses in at least one subgraph from their drive. Thirty eight of the participants could identify why email addresses were clustered in their "useful" subgraph. When the clustering was identified during the interview, the participants could further label the clusters as family, work, friends, or possibly a time period in their life.

The identification of "useful" subgraphs followed the methodology described in Section 4.4. Commonly on a "useful" subgraph, the degree and eigenvector centrality identified the owner of the drive. The owner of the drive would have the highest degree and an eigenvector centrality score of 1.0 or close to 1.0. Visually, this would put the drive owner at the center of the subgraph after we ran the Fruchterman Reingold layout in Gephi, shown in Figure 5.1 and in Figure 5.2. Another common feature of social networks was the clustering of groups of email addresses with a common domain or social group. An example showing color coded clustering of a participant's social network with clear separation between social groups using modularity classes is in Figure 5.1.

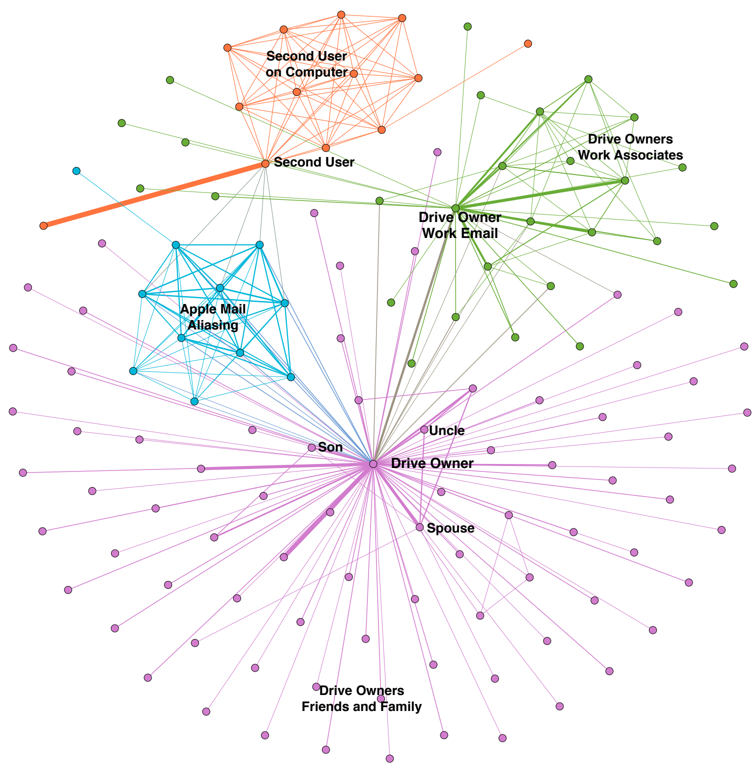


Figure 5.1. Disk02c7 showing an unfiltered subgraph with clustering of different social groups in the participant's social network. The different colors of nodes and edges represent different modularity classes, which clustered different social groups. The layout is Fruchterman Reingold.

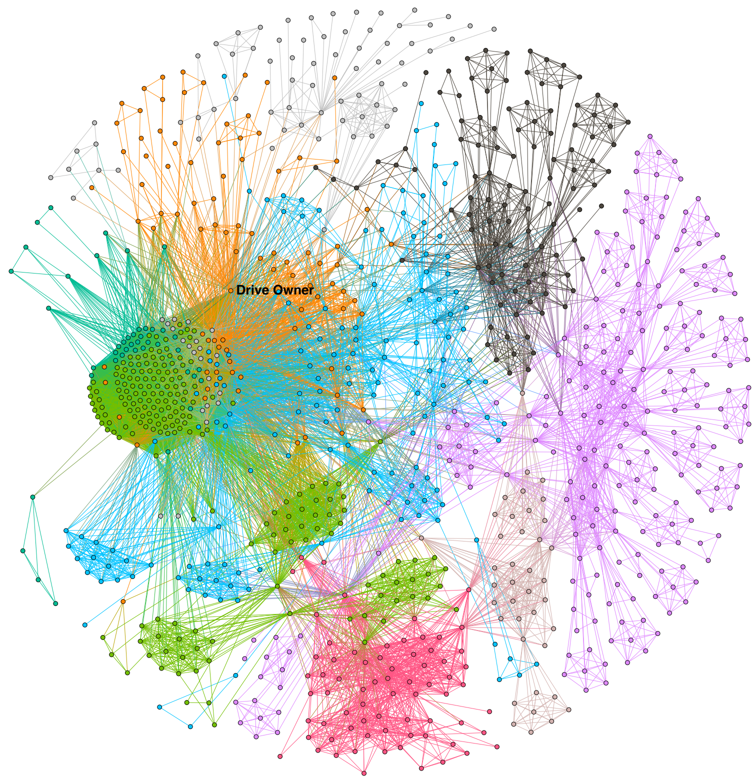


Figure 5.2. Disk04c2 showing an unfiltered subgraph with clustering of different social groups in the participant's social network. The different colors of nodes and edges represent different modularity classes which corresponded to different social groups. The layout is Fruchterman Reingold.

### 5.3.1 Useful Subgraphs Containing Noise

Noise, in our experiment, is defined as the unwanted email addresses in the "useful" subgraph that mask email addresses in the social network. Examples of noise in subgraphs consisted of email addresses representing distribution lists and a "not useful"subgraph embedded in the "useful" subgraph. We encountered both of these types of noise during the experiment. We describe each in detail below.

In addition, we propose a noise-removal method that relies on filtering by modularity class since each class is a cluster of related emails. The same method to remove the noise is implemented on each subgraph we identified having noise, such as distribution lists or embedded "not useful" subgraphs. After the communities are identified and numbered by modularity class , that modularity class is filtered out of the subgraph.

**Distribution Lists**

A distribution list is a common way for a single user to send the same email to many recipients without manually sending the same email multiple times. The distribution lists found in our dataset are related to communications from organizations the participants are associated with, such as the Naval Postgraduate School and various military organization groups.

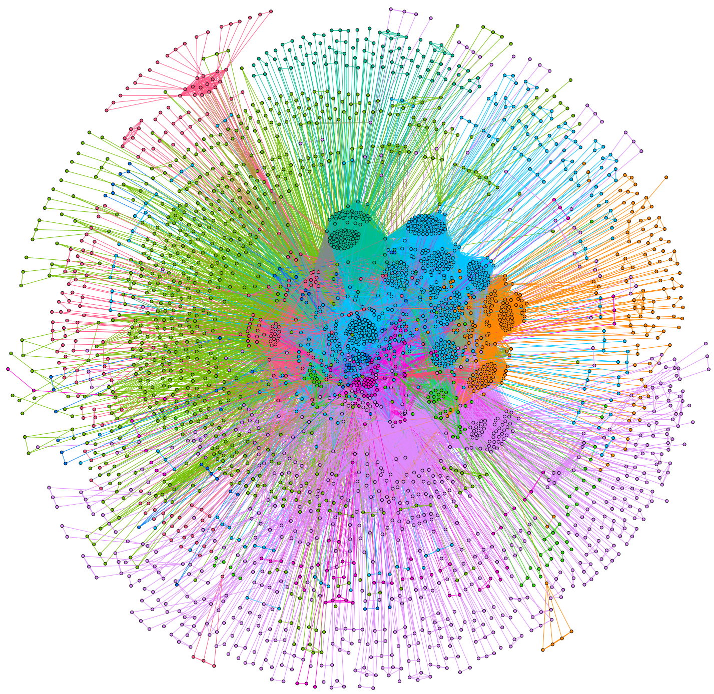

Figure 5.3. The social network of disk03c1 demonstrates a subgraph containing several distribution lists the drive owner was a part of at NPS. These are obscuring the clusters containing useful communications. The layout used for this subgraph is Fruchterman Reingold.

Examples of the noise added by distribution lists are shown in Figure 5.3, Figure 5.5, and Figure 5.4. In Figure 5.3 we show the social network found on disk03c1 and in Figure 5.5 we show the filtered subgraph with distribution clusters removed. In Figure 5.4, we show the distribution list clusters from the disk03c1 subgraph.

In these subgraphs, the different color clustered groups represent different distribution lists the drive owner was a part of at the Naval Postgraduate School and other military organizations. The distribution clusters create large amount of noise within a subgraph obscuring useful email communications. The method we used to remove this noise, once the participant identified the distribution lists, was filtering out the distribution lists by removing all the nodes in the modularity class identified by the user. This approach commonly exposed the useful social network.

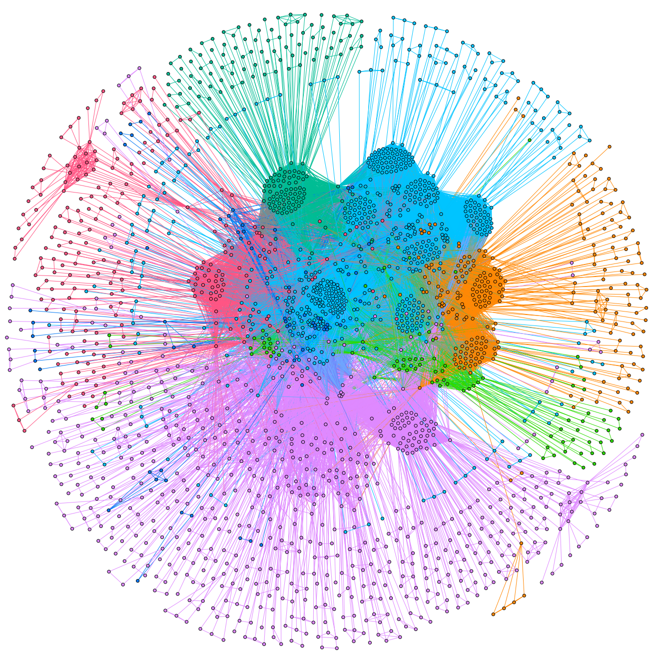

Figure 5.4. The distribution lists in the disk03c1 subgraph without the two useful clusters of the drive owners social network. This graph shows the noise that was removed to reveal the useful communications. It contains are $2,188$ nodes and $57,647$ edges out of the total $22,311$ nodes and $1,823,655$ edges shown in Figure 5.3. The layout is Fruchterman Reingold.

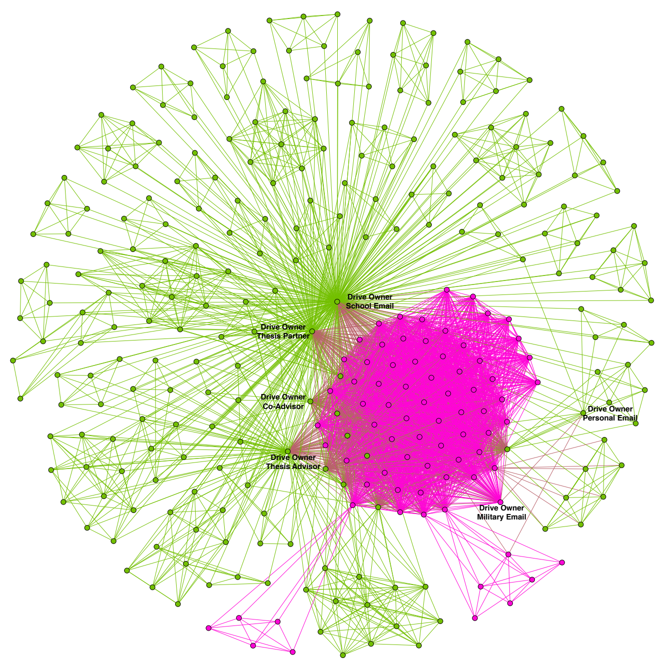


Figure 5.5. The social network found on disk03c1 using filtering with modularity class. The two modularity classes in the subgraph contain the drive owner's email addresses. Other labels applied in the figure are of individuals the drive owner communicated with frequently. The Fruchterman Reingold layout is used on this subgraph.

### Embedded "Not Useful" Subgraph

We characterize a component of a useful subgraph as an *embedded "not useful" subgraph* when it comprises a "not useful" subgraph that we have encountered separately in our dataset. An example of this is the subgraph we categorized as the "travel" subgraph. This set of nodes and edges appeared three times in our dataset as a separate subgraph, and in each of these occurrences it was not adjacent to an email address of the participant.

However, two other times it was located in the “useful” subgraph of the participant. The “travel” subgraph added approximately 580 nodes and $165,000$ edges of noise to a “useful” subgraph. The “travel” subgraph is another case where filtering a whole modularity class assisted in minimizing noise in the subgraph containing the social network.

In Figure 5.6, Figure 5.7, and Figure 5.8, we show how removing the embedded “not useful” subgraph reduces the number of nodes and edges that needed to be analyzed. In both cases, the embedded subgraph was connected to the rest of the subgraph by only two email addresses, each of which is identified in Figure 5.6.

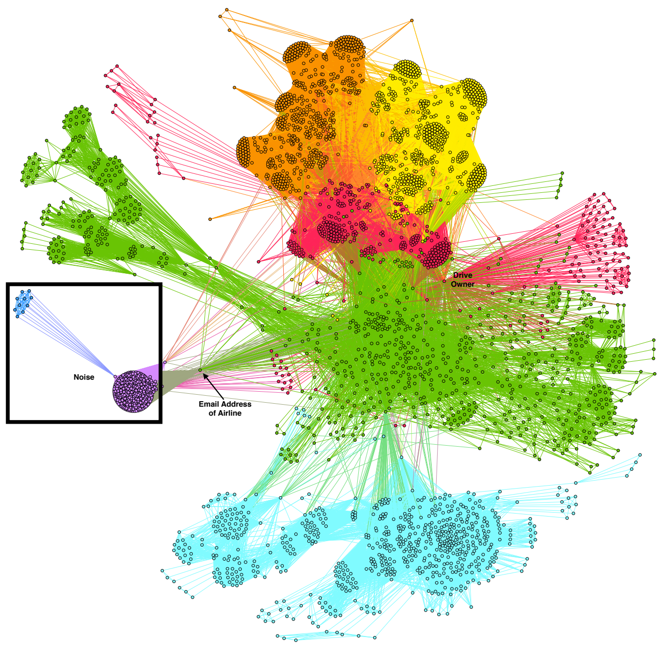


Figure 5.6. The social network found on disk32c2 showing clusters of communication by the drive owner. The clusters include personal and professional email communications. The highlighted box in the figure is showing noise, or "not useful" nodes and edges. The layout used for this figure is Fruchterman Reingold.This subgraph contains $5,220$ nodes and $316,885$ edges.

There are two different nodes that connect the drive owner to the "travel" modularity classes (see Section 5.3.2 for further discussion on the "travel" subgraph) shown in Figure 5.7 and then in Figure 5.5.

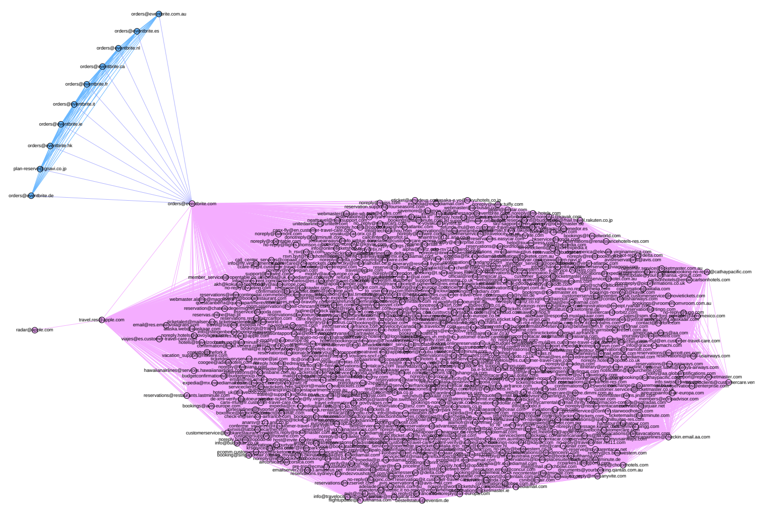

Figure 5.7. The extracted modularity classes showing the "travel" subgraph embedded in the "useful" subgraph. The layout used after we exported the two modularity classes is Fruchterman Reingold. We also used a label adjustment function to make the email addresses visible. There are 584 nodes and 163, 934 edges in this "not useful" portion of disk31c1 subgraph.

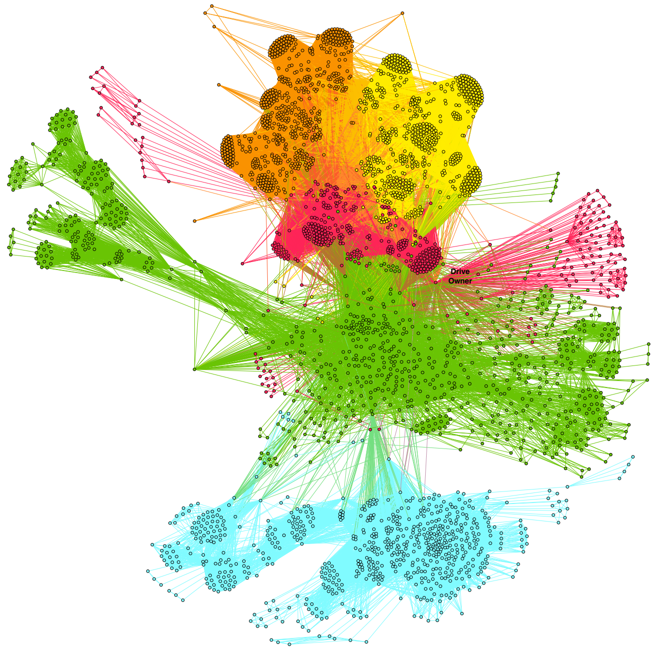


Figure 5.8. The social network without the "travel" portion of the subgraph. After removing the noise, this subgraph has $3,232$ nodes and $150,498$ edges. The layout is Fruchterman Reingold.

### 5.3.2 "Not Useful" Subgraphs

The majority of the subgraphs are not useful for identifying social networks on a drive. However, it is important to classify these subgraphs into categories to provide test and training data that could be used in future work to build a machine-learning classifier. Frequently, software installations on the participant's computer create their own subgraph on the drive. Because of this behavior, we identified several subgraphs that occurred on multiple drives. We verified the software package or location of the email addresses on the drives using identify_filename.py.

In this subsection, we characterize common "not useful" subgraphs and explain why we concluded the subgraphs are "not useful" for building the participant's social network.

**Graphics Subgraphs**

We labeled the most common "not useful" subgraph "graphics." This subgraph contained the file names of Portable Network Graphics (PNG) files and JPEG files. An example of the PNG file name format was *<name>@2x.png*, following a common convention for naming graphics files that need to be stored at multiple resolutions for compatibility with different display settings.

We found that bulk_extractor truncates the last letter and mistakes these file names for email addresses using the .pn top-level domain, a relatively uncommon country-code top-level domain assigned to the Pitcairn Islands. As a result, these file names with the final "g" omitted were added to the email.txt file.

We used identify_filename.py (see Section 2.5) to find the location of the file names and verified the files were images used in applications. The context field in email.txt verified that the character after "pn" was the letter "g." Modifying bulk_extractor to check if the characters "pn" are followed by a "g" would provide output with less noise.

In our experiment, this bug created 102 "not useful" subgraphs. The 102 subgraphs added $33,912$ nodes and $1,105,342$ edges of noise to our dataset. The "graphics" subgraphs all look slightly different but contain the same ".pn" at the end of the node labels. Two examples of the "graphics" subgraphs are shown in Figure 5.9.

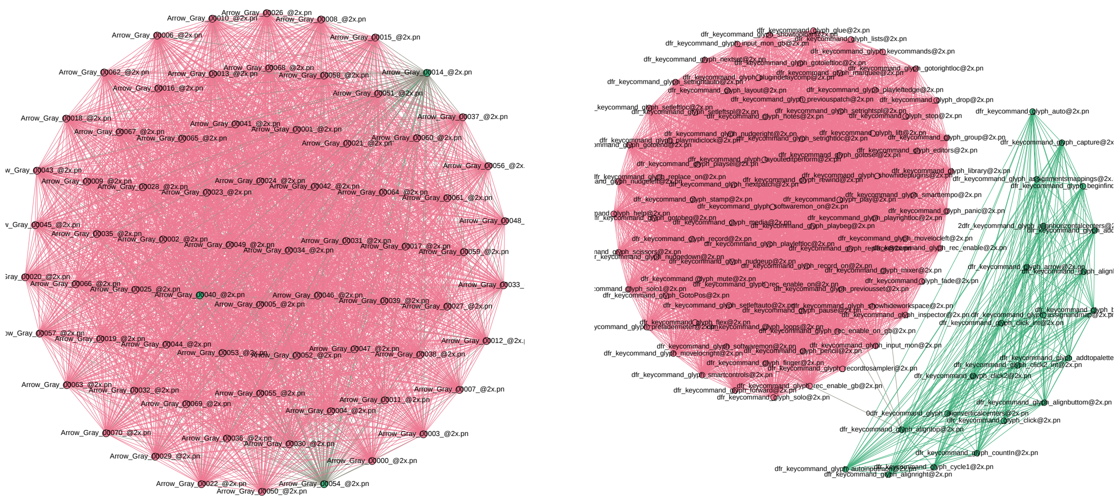

Figure 5.9. "Graphics" subgraphs of PNG file names using Fruchterman Reingold layout. The subgraph to the left, with Arrow_Gray node labels, is an completely connected subgraph. The graph to the right shows two modularity classes created by Gephi's modularity statistics function. Both graphs represent icons used by an application on the drive.

**Virtual Machines Subgraphs**

A question we verified with participants after looking at their subgraphs was if they used virtual machines (VM) on their computer. Twenty seven users stated they used some variation of virtual machines on their computers.

We found 27 corresponding subgraphs that contained email addresses associated with virtual machines stored on the drives. We categorized these as "VM" subgraphs. The number of nodes in these subgraphs ranged from 8,000 to 15,000 nodes. On drives where a VM was present, the "VM" subgraph had the highest node count. The presence of the following email addresses, associated with either the highest degree or highest eigenvector centrality, was a strong indicator that the subgraph was derived from a virtual machine:

- ubuntu-devel-discuss@lists-unbuntu.com
- <name>@thewrittenword.com
- <name>@twinsun.com

Another characteristic of the "VM" subgraph was clusters of email addresses with the domains listed below. For example, one cluster might contain several <name>@redhat.org addresses, and another might contain several <name>@debian.org email addresses. If the address with the highest degree and eigenvector centrality was one of the addresses listed above, we would check for these clusters of emails from the following domains:

- <name>@debian.org
- <name>@redhat.org
- <name>@ubuntu.com
- <name>@gnome.org

During the interview, the participants used the search function in Gephi to manually verify there were no useful email addresses in the "VM " subgraph. Out of the 27 VM subgraphs, there were three instances in which the participant found useful email addresses in the subgraph. In these cases the subgraph was labeled "useful" and also categorized as the "VM" subgraph. Filtering by modularity class, we were able to visualize the participant's social network by removing the "VM" clusters. This situation occurred in cases where the participant would access their email from inside the VM.

**Microsoft Office Subgraphs**

The installation of Microsoft Office created a subgraph on the drive. We identified the "Microsoft" subgraph on 22 drives. The subgraph contains 277 nodes, $6,971$ edges, and has an average degree of 50.3321. The email addresses in the "Microsoft" subgraph are found at the following path:

*Program Files (x86)/Microsoft Office/root/Office16/mset7db.kic*

The mset7db.kic file is a database file. Using Microsoft Access, we verified the email addresses in this subgraph were stored in the mset7db.kic file.

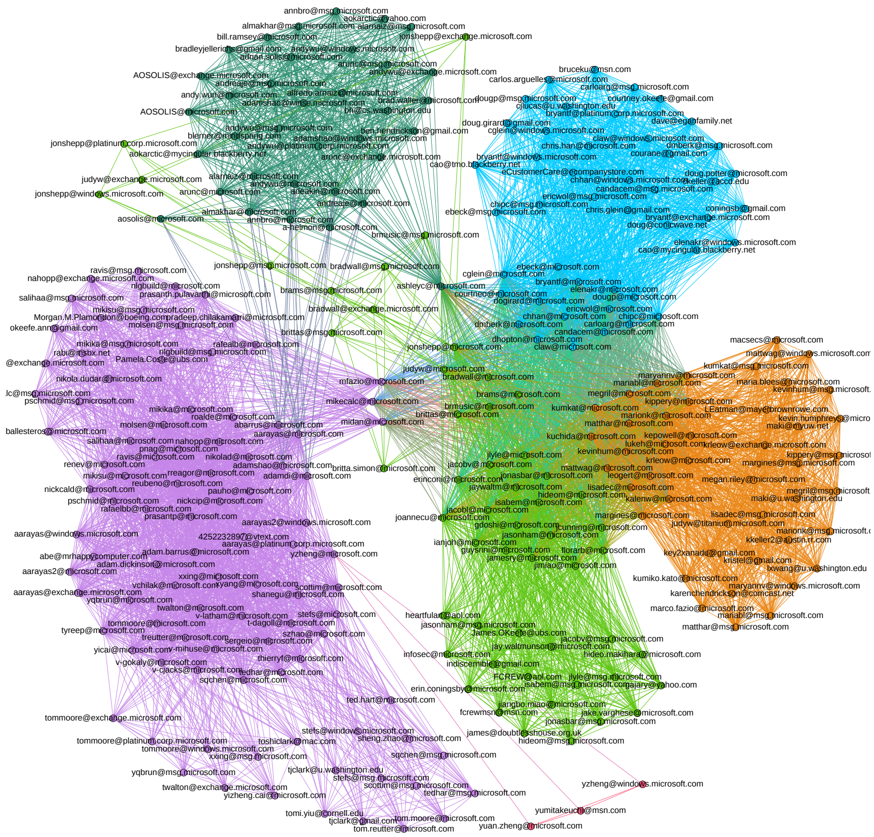

Figure 5.10. Microsoft Office installation from disk01c1 using Fruchterman Reingold layout. There are six modularity classes in this subgraph represented by different colors.

**Libraries: Libsamplerate and Libsndfile Subgraphs**

The "libsamplerate" subgraph is in the dataset two times and "libsndfile" subgraph is in the dataset 17 times. The library "libsamplerate" converts audio from one sample rate to another [50]. The library "libsndfile" is "a C library for reading and writing files containing sampled sounds through one standard library interface" [51]. The ".so" extension is used for C and C++ shared libraries [52]. Similar to the PNG file names, ".so" is a top level domain for the country of Somalia, which causes bulk_extractor to mistake it for an email address. Correcting this bug would remove these addresses prior to the subgraph construction process.

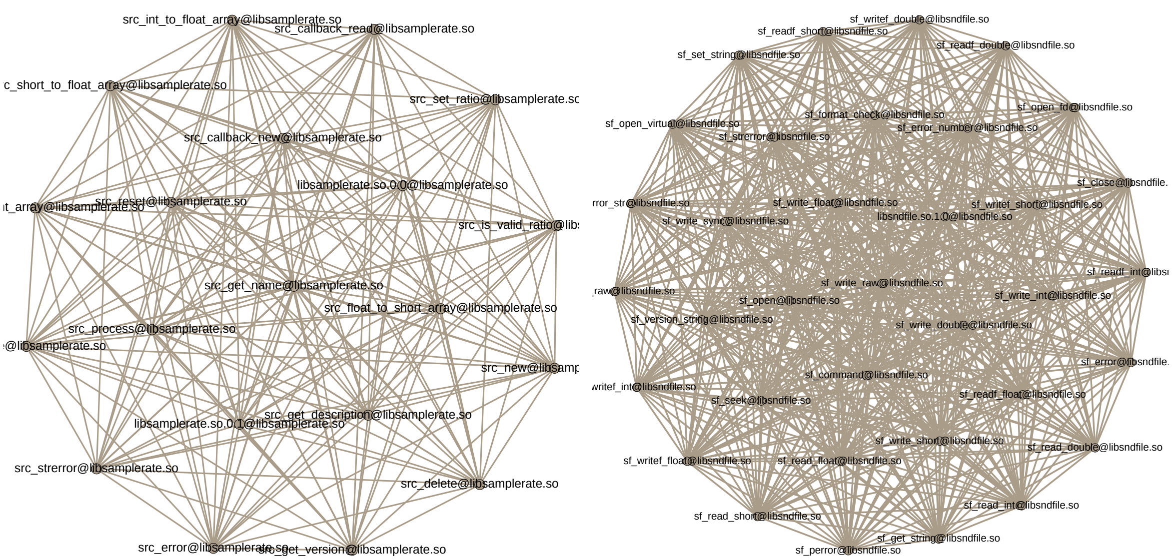


Figure 5.11. "Libsamplerate" and "Libsndfile" sugbraphs are completely connected. The Fruchterman Reingold layout was used to produce these graphs.

**Travel Subgraphs**

The subgraph we categorized as "travel" contained several email addresses of airlines, hotels, and entertainment companies associated with travel or events, such as Expedia, Avis, US Airways, and Ticketmaster. We see this subgraph five times in our dataset — three times as a separate subgraph and twice as an "embedded subgraph inside a "useful" subgraph. As mentioned earlier in the section, the "travel" nodes and edges cause the actual social network subgraph of the participant to be filled with noise and require additional analysis. The location of the subgraph's email addresses on the drive is:

```
System/Library/Assets/com_apple_MobileAsset_CoreSuggestions/
<drive_specific_characters>.asset/AssetData/ReverseTemplateJS/main.js
```

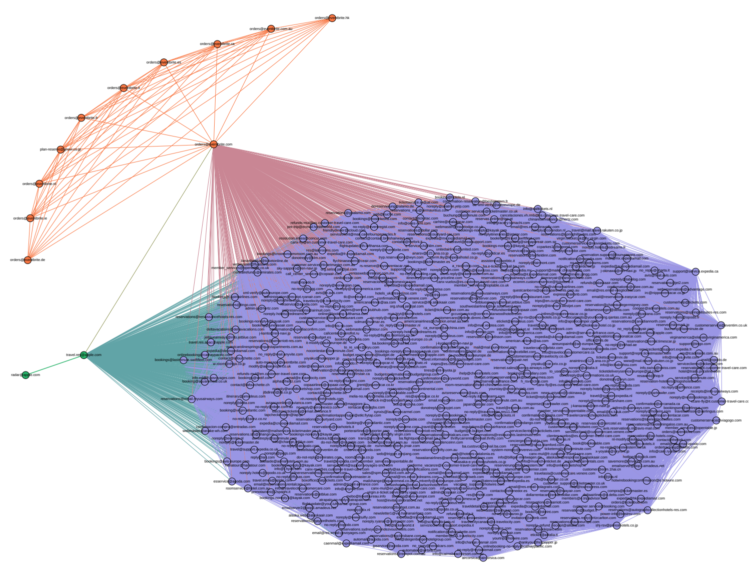

Figure 5.12. The "travel" subgraph on disk08. All of the clusters in this subgraph connect to orders@eventbrite.com. The orange cluster contains different domains for orders@eventbrite connecting to orders@eventbrite.com. This subgraph is similar to the extracted nodes and edges from disk31c2 in Figure 5.7.

### The "Cert" Subgraph

The "cert" subgraph is an example of a "not useful" subgraph. Using identify_filename.py to find the location of these email addresses shows that they are stored in the file msg_43.txt, a test file associated with the Python email module, which is designed for creating, parsing and sending email addresses using Python. These email addresses all have the format "xxxxxxxxx@dot.ca.gov" and are part of the list of recipients in the message included in the test file.

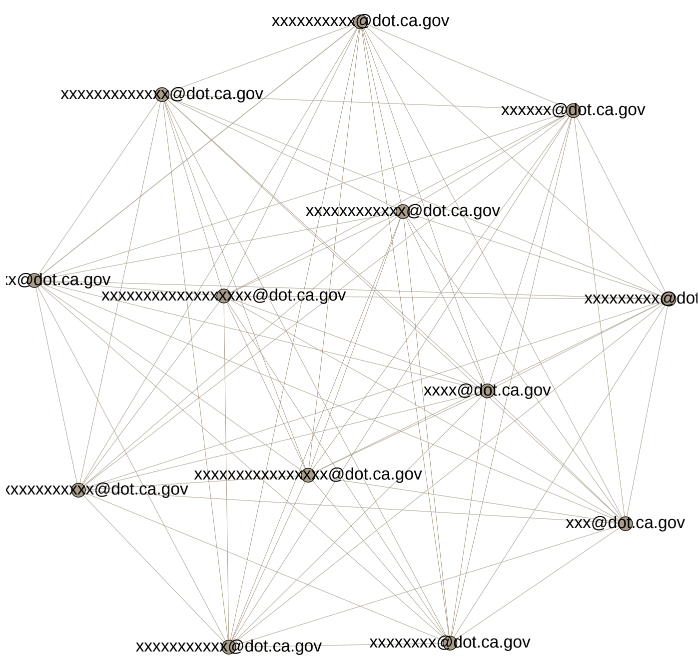


Figure 5.13. "Cert" subgraph is a completely connected subgraph. These email addresses are found in a text file on the drive.

**Email Addresses in File**

In our experiment, we found several completely connected subgraphs that resulted from email addresses found in Microsoft Excel spreadsheets. We found the spreadsheets that emails were stored in using identify_filename.py. When email addresses are found in documents, the nodes in these subgraphs are completely connected because all of the email addresses are within the 128-byte window used by friend.py.

## 5.4 Unusual Results

During the experiment, we encountered a few drives that required additional analysis. We initially used identify_filename.py to assist in answering questions derived from the participant interviews. Three participants' social networks contained a portion of their name in the local part of several email addresses which the user did not recognize Explaining the presence of these email addresses required finding where the addresses were stored on the drive. Another unexplained outcome in a subgraph was the pattern of email addresses in a "chain" in the participant's social network.

### 5.4.1 Portion of Drive Owner Name as Email Address

In the a subgraph that consisted of email addresses containing a portion of the participant's name, the format of the email address was <*name of participant*>@<*several different domains or websites*>. For example, if the normal log in to a website was johndoe, the email address would show up as doe@<some_website.com>. We would commonly expect to see johndoe@<some_website.com> and not just a portion of the name. During the interviews, the participant verified that their email addresses contained numbers or other letters that were not found in the subgraph. After using identify_filename.py, we found the email addresses to be stored with cached websites. The name in the email address is from the name of the profile the participant uses and not their email address. The email addresses are stored on the drive at:

```
Users/<name>/AppData/Local/Microsoft/Windows/WebCache/WebCacheV01.dat
```

### 5.4.2 AutoComplete in Microsoft

Several subgraphs contained an extended series of nodes connected in a chain. An example of this behavior is found in Figure 5.15. In the figure, the nodes around the outside of the subgraph are chained together. After seeing this behavior occur in more than one subgraph, we used identify_filename.py to find the location of the email addresses. The purpose of this file is to store names used by the AutoComplete and name checking features in Microsoft [53].The email addresses are located under the profile of the user at:

```
$/AppData/Local/Microsoft/Outlook/RoamCache/Stream
\_autocomple\_<list of numbers and letters>.dat$
```

We initially used the Fruchterman Reingold layout shown in Figure 5.14. However, we found the Force Atlas 2 layout better represented the behavior of the Microsoft AutoComplete lists when visualized in Gephi. This layout is shown in Figure 5.15.

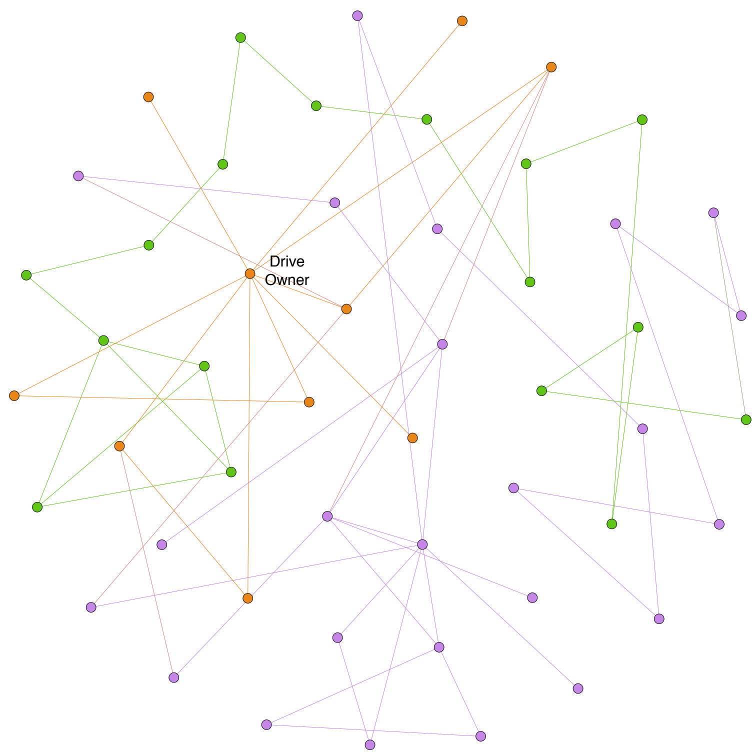


Figure 5.14. Microsoft AutoComplete Name List on disk09c6 using the Fruchterman Reingold layout. Of the three modularity classes shown, the orange cluster contains the drive owner. The green and purple clusters are chained together but visualizing this chain of email addresses is difficult with this layout.

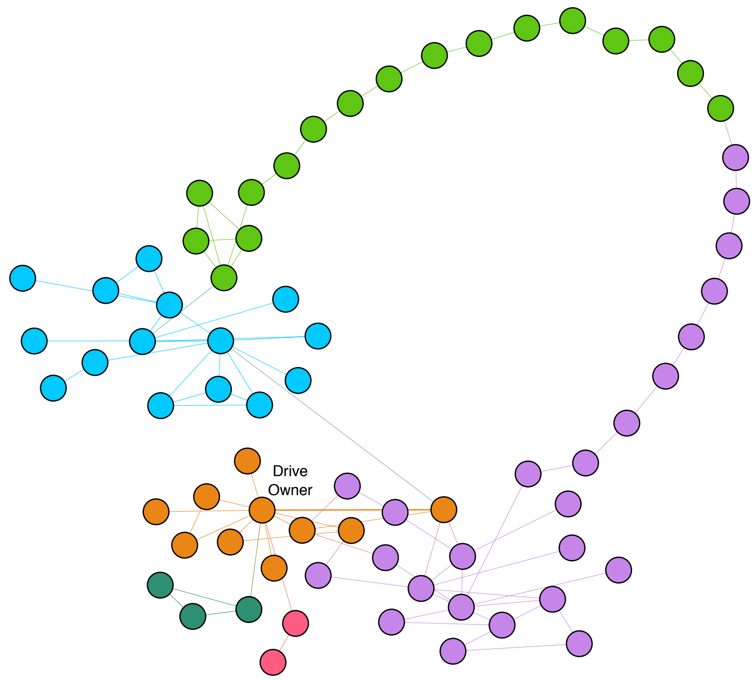


Figure 5.15. Microsoft AutoComplete Name List on disk09c6 using the Force Atlas 2 layout. Of the modularity classes shown, the orange cluster contains the drive owner. The green and purple clusters are chained together and contain email addresses found in the Microsoft Outlook folder discussed above.

**Extremely Large Subgraph**

One subgraph created was over two gigabytes in size. In comparison, the majority of the GEXF files are kilobytes in size. We found this was a file containing email addresses from an organizational directory of the National Parks Service. The subgraph has $12,668$ nodes and $75,948,507$ edges with an average degree of $11,990$. The large file size prevented visualization with Gephi. We analyzed the GEXF file through a text editor during the interview. This file and others found on the same drive are examples of email addresses found in a document, which produces "not useful" subgraphs.

# CHAPTER 6: Conclusion and Future Work

This final chapter presents our conclusions. We will answer the question posed in our initial chapter and present some suggestions for future work.

## 6.1 Conclusion

The motivation behind our work was to reduce the time a forensic analyst has to spend manually examining data to find email addresses that directly contribute to developing the user's social network. Automating steps in this process will allow the analyst to focus on the data that is most relevant to building the social network. Our research addresses this motivating factor by testing a new algorithm for identifying connections between email addresses found on the user's drive.

### 6.1.1 Identifying User Communication Patterns

*Does our new algorithm for constructing social networks improve our ability to correctly model user communication patterns?*

Yes. In order to adequately test our new algorithm, we built an 11 terabyte dataset (over five times larger than that which was used to test the previous algorithm). We successfully acquired and analyzed 45 participant drives. Using our graph-labeling methodology, we achieved a recall of 0.9677, precision of 0.6316, and F-score of 0.7643. We were also able to identify specific groups within the user's social network. These included college, work, sports, family, friends, and business associates.

### 6.1.2 Proposed Improvements to Bulk Extractor

As noted in Chapter 5, bulk_extractor's email.txt feature file provides numerous artifacts that are not email addresses but were formatted in a similar manner. A very common artifact that was mischaracterized as an email address was an image file (.png). The bulk_extractor program would leave the "g" off and list what it thought was an email address as "example.pn". These false email addresses accounted for 102 out of 450 graphs.

If bulk_extractor was updated to exclude files that contained the .png suffix, there would be much less noise and fewer "not useful" graphs. Another improvement would be to prevent the creation of a separate email address in the feature file when the only difference is the case of the characters. This would reduce noise, would create additional edges for current nodes and could potentially affect the modularity class communities.

## 6.2 Future Work

While our research provided an answer to our question and contributed to our goal of automation, there are several areas for further research and development. These include comparing the impact of solid state storage versus magnetic spinning disks, applying this method to other devices and artifacts beyond email addresses, and automatic classification and testing with SVMs.

### 6.2.1 Solid State Drive and Magnetic Hard Drive Comparison

Based on our research and understanding of SSD operation, there should be less recoverable data on the SSD. However, our research did not show any noticeable difference in the data recovered. This suggests the need for additional testing in a controlled environment to compare data recovery between the two types of drives.

### 6.2.2 Extending the Other Devices Types and Applications

Our results were conducted solely on hard drives from laptop computers. Since communication trends have expanded to various applications on more portable devices, future work with cell phones and tablets would be beneficial. We encountered graphs associated with other communication applications such as Whatsapp however, these type of applications are much more prevalent on cell phones.

### 6.2.3 Classification with Support Vector Machines

During classification of the subgraphs, we looked in to fingerprinting the subgraphs to better classify the subgraphs into categories for use with SVMs. We found that subgraphs with the same node count, same edge count, and were completely connected, would classify the same even though they are not isomorphic. The algorithm we researched was GFP.py (see Section 3.5.1 to create a fingerprint of each graph and using Canberra distance (see Section 3.5.1 to calculate how similar the subgraphs are to each other. The feature vectors in GFP.py do not take into account the weight of the edges in the subgraph. We discovered examples of this in our data set. Adding weights to the functions in the GFP algorithm would differentiate between graphs that have the same node and edge count. This future work could improve classification of the subgraphs for use with SVMs.

During the process of manually labeling our subgraphs, we looked into graph "fingerprinting" techniques as a method of efficiently identifying structural similarities between subgraphs. The algorithm we researched was Graph Fingerprint Comparison, or GFPC (see Section 3.5.1). This algorithm creates a fingerprint of each graph and uses Canberra distance (see Section 3.5.1) to calculate the similarity between subgraphs. Because fingerprints are represented as feature vectors, we hypothesized that they could be used to train a support vector machine classifier. However, in their current implementation, graph fingerprints do not take into account the weight of the edges in the subgraph. As a result, we found that completely connected subgraphs with the same node count and same edge count would be treated as identical even when they were not isomorphic. We discovered examples of this in our dataset. This problem could be addressed by modifying the features used to create the graph prints so they incorporate edge weight. This work could potentially contribute to the creation of an effective SVM classifier that automatically identifies useful graphs.

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX A: Friend.py

```
#!/usr/bin/env python3

# Program: friend.py
# Author: Michael McCarrin <mrmccarr@nps.edu>
# Description: Produce graphs from bulk_extractor features.

import networkx as nx
import sys
import os
import argparse
import timeit
import collections
from concurrent.futures import ProcessPoolExecutor, as_completed
from os.path import join, basename, normpath

def ingest_feature(be_dir, feature_file):
  list_dict = {}
  aliases = {}
  email_file = join(be_dir, feature_file)

  with open(email_file, "rb") as f:
    for l in f:
      try:
        line = l.decode("utf-8")
      except:
        print("Line contains invalid utf-8; skipping")

      if line[0] == "#":
        continue

      raw_offset, raw_alias, remainder = line.split("\t", 2)
```

```
      alias  = raw_alias.replace("\\x00","")
      aliases[alias] = {'context': remainder}

      if raw_offset.isdigit():
        root   = "top"
        offset = int(raw_offset)
      else:
        complex_offset = raw_offset.split("-")
        root   = "-".join(complex_offset[:-1])
        offset = int(complex_offset[-1])

      if root not in list_dict:
        list_dict[root] = [(offset, alias)]
      else:
        list_dict[root].append((offset,alias))

    for l in list_dict.values():
      l.sort()

  return aliases, list_dict

def nx_weighted_edge_list(edge_dict):
  w_edges = []
  for p, w in edge_dict.items():
    w_edges.append((p[0], p[1], {"weight":w}))
  return w_edges

def fixed_find_edges(feature_data, win_size=256):
  node_q = collections.deque()
  edges  = collections.defaultdict(int)

  for l in feature_data.values():
    node_q.clear()
    for offset, alias in l:
```

```
      while (node_q and (node_q[0][0] < offset - win_size)):
        node_q.popleft()

      for n in node_q:
        pair = tuple(sorted([alias, n[1]]))
        if pair[0] != pair[1]:
          edges[pair] += 1

      node_q.append((offset, alias))

  weighted_edges = nx_weighted_edge_list(edges)

  return edges, weighted_edges

def stretchy_find_edges(feature_data, win_size=256):
  node_q = collections.deque()
  edges  = collections.defaultdict(int)

  for feature_list in feature_data.values():
    node_q.clear()
    for offset, alias in feature_list:
      while (node_q and (node_q[-1][0] < offset - win_size)):
        node_q.popleft()

      for n in node_q:
        pair = tuple(sorted([alias, n[1]]))
        if pair[0] != pair[1]:
          edges[pair] += 1

      node_q.append((offset, alias))

  weighted_edges = nx_weighted_edge_list(edges)

  return edges, weighted_edges
```

```
def make_gephi(V, E):
  G = nx.Graph()
  G.add_nodes_from(V)
  G.add_edges_from(E)
  return G

def find_big_connected_components(G, num=5):
  n = nx.number_of_nodes(G)
  con_comps  = list(sorted(nx.connected_component_subgraphs(G),
    key=len, reverse=True))
  return con_comps[:num]

class Summary:
  def __init__(self, path):
    self.lines = []
    self.path  = path

  def print_last_n(self, n=0):
    index = 0 if n == 0 else len(self.lines) - n
    for l in self.lines[index:]: print(l)

  def write(self):
    with open(self.path, "w") as f:
      for l in self.lines: print(l, file=f)

  def add(self, line):
    self.lines.append(line)

def get_dirs(dir_list):
  dirs = []
  with open(dir_list, "r") as d:
    for line in d:
      dirs.append(line.strip().rstrip(os.sep))
  return dirs
```

```
def process_many(in_dirs, num_jobs, outdir, args):
  out_dirs  = [join(outdir, basename(subdir)) for subdir in in_dirs]
  arguments = zip(in_dirs, out_dirs, [args]*num_jobs, [True]*num_jobs)
  with ProcessPoolExecutor() as executor:
    executor.map(process_one, arguments)


def process_one(arguments):
  be_dir, out_dir, args, multi_mode = arguments
  bar      = "="*80
  win_size = args.win_size
  loners = args.keep_loners
  big_subs = args.big_subgraphs
  feature_file = args.feature_file

  if args.fixed_window:
    find_edges = fixed_find_edges
  else:
    find_edges = stretchy_find_edges
  #For GEXF Files
  gname    = basename(normpath(be_dir)) + "_" + str(win_size) + ".gexf"
  #For Graphml Files
  #gname    = basename(normpath(be_dir)) + "_" + str(win_size) + ".graphml"

  sumfile  = basename(normpath(be_dir)) + "_" + str(win_size) + "_summary.txt"

  if not os.path.exists(out_dir):
    os.makedirs(out_dir)

  sumpath  = join(out_dir, sumfile)
  gpath    = join(out_dir, gname)
  summary  = Summary(sumpath)
  summary.add("Friend summary for command line invocation:")
```

```
summary.add(" ".join(sys.argv))
summary.add(bar)
summary.add("Using Window Size: {}".format(win_size))
summary.add("Writing graphs to: {}".format(gpath))
summary.add(bar)
if not multi_mode: summary.print_last_n()

start             = timeit.default_timer()
nodes, feature_data = ingest_feature(be_dir, feature_file)
ingest_time       = timeit.default_timer()

summary.add("Ingested in {} seconds".format(ingest_time - start))
if not multi_mode: summary.print_last_n(1)

edges, weighted_edges = find_edges(feature_data, win_size)
edge_time             = timeit.default_timer()

summary.add("All neighbors linked in {} seconds".format(edge_time - ingest_time))
if not multi_mode: summary.print_last_n(1)
if not loners:
  friends = {}
  for e in edges.keys():
    friends[e[0]] = nodes[e[0]]
    friends[e[1]] = nodes[e[1]]
  nodes = friends

G = make_gephi(nodes.items(),weighted_edges)
#GEXF Output
nx.write_gexf(G, gpath)
#Graphml Output
#nx.write_graphml(G, gpath)
end = timeit.default_timer()

summary.add("Finished in {} seconds".format(end - start))
if not multi_mode: summary.print_last_n(1)
```

```
summary.add(bar)
summary.add("Summary Statistics for Whole Graph:")
summary.add(bar)
summary.add(nx.info(G))

if big_subs:
  subgraph_start = timeit.default_timer()
  i = 1
  summary.add("Looking for the top {} subgraphs".format(big_subs))
  if not multi_mode: summary.print_last_n(1)

  major_subs = find_big_connected_components(G, big_subs)
  subgraph_end = timeit.default_timer()
  subgraph_time = subgraph_end - subgraph_start

  summary.add("Found {} big subgraphs in {} seconds.".format(len(major_subs)
      , subgraph_time))
  if not multi_mode: summary.print_last_n(1)

  for g in major_subs:
    summary.add(bar)
    summary.add("Summary Statistics for subgraph {}:".format(i))
    summary.add(bar)
    summary.add(nx.info(g))
    #GEXF
    gname = basename(normpath(be_dir)) + "_" + str(win_size) +
      "sub" + str(i) + ".gexf"
    #Graphml
    #gname = basename(normpath(be_dir)) + "_" + str(win_size) +
      "sub" + str(i) + ".graphml"
    out   = join(out_dir, gname)
    #GEXF
    nx.write_gexf(g, out)
    #Graphml
    nx.write_graphml(g, out)
```

```
        i += 1

    summary.write()

    if multi_mode:
      print(".", end="")
      sys.stdout.flush()
    else:
      print("Check {} for a summary of the results.".format(sumpath))

if __name__=="__main__":
    parser = argparse.ArgumentParser(description='Discover networks
    in bulk extractor email.txt output.')

    group = parser.add_mutually_exclusive_group(required=True)

    group.add_argument('be_results_path', nargs='?',
      help="Path to bulk_extractor output files. Required unless
      running with the -p option.")

    parser.add_argument('-o', '--output-path', required=True,
      help="Sets the directory where befriend should store its output (required).")

    parser.add_argument('-w', '--win-size', type=int, default=128,
      help='Optionally sets the window size in bytes. Default is 128.')

    parser.add_argument('-f', '--feature-file', default="email.txt",
      help='Sets the bulk_extractor feature file to be used for analysis.
      The default is email.txt.
      Any other value is experimental.')

    parser.add_argument('-F', '--fixed_window', action='store_true', default=False,
      help='Only link addresses found within the distance defined by win_size.
      This was the old default behavior.
      This is probably not what you want, so it is not recommended
```

```
  to use this option unless you want to reproduce legacy behavior.')

parser.add_argument('-k', '--keep-loners', action='store_true', default=False,
  help='Keep all nodes, even those with no edges to other nodes.')

parser.add_argument('-m', '--big-subgraphs', metavar='N', type=int, default=10,
  help='Save a separate gexf file for the top N largest connected
  components of the graph. Default is 10.')

group.add_argument('-p', '--parallel', dest="dir_list",
  help='Specify name of a file containing a list of bulk extractor
  output directories and process these in parallel.
  An output directory will be created for each as a subdirectory
  of the output path.')

args     = parser.parse_args()

if args.dir_list:
  multi_start = timeit.default_timer()
  in_dirs   = get_dirs(args.dir_list)
  num_jobs  = len(in_dirs)
  process_many(in_dirs, num_jobs, args.output_path, args)
  very_end = timeit.default_timer()
  print()
  print("Completed {} jobs in {} seconds.".format(num_jobs, very_end - multi_start))
else:
  process_one((args.be_results_path, args.output_path, args, False))
```

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX B: Ground Truth Interview Questions for Drive Owners

The interview questions listed in this appendix are approved IRB questions used in Green's [3] thesis. However, we modified some of the questions with IRB approval.

Ground Truth Interview Questions for Drive Owners

Questions The following are questions which were asked during exit-interviews with those who volunteered their storage media:

1. You will be presented with a spreadsheet of email addresses and images of several graphs. For each graph, indicate whether you recognize the email addresses in the subgraph.

2. If you recognize an email addresses, are these addresses of a people with whom you communicate by email?

3. In general, does the graph accurately describe the relationship between the correspondents listed?

4. If the graph is clustered into different groups, are you able to discern a reason for the clustering (for example, different groups may represent coworkers, family, friends, etc.)?

5. Why do you believe that these email addresses were collected as pairs? Are these emails often on the same address lines for emails or some other situation such as used for log on purposes or all located within the same webpage?

6. What operating system do you use (e.g. Windows 7, Windows 8, OSX 10.7, etc.)?

7. Is your drive solid state drive?

8. What email client do you use (Outlook, Outlook Express, Apple Mail, Web-based email, other (please describe))? If you use multiple clients (for example if you use Outlook for official email and Gmail for personal email), please list all of them and approximate the percent of time used on each.

9. How long have you used this computer?

10. Have there been any prior users / owners of this computer? Do you share the computer with anyone else?

11. How often do you communicate by email (daily / weekly / monthly)?

12. Is there anything else you would like to add? For example, if the analysis you were presented with is incorrect, do you have an explanation about what went wrong?

# APPENDIX C:
# Aggregated Hardware and Software Components of Data Set

Table C.1

| Disk Name | Brand of Computer | Drive Type | Size of Drive | Age of Drive(yrs) | Operating System | Email Provider | Email Access Method | Shared Computer | Communication Frequency | VM Software |
|---|---|---|---|---|---|---|---|---|---|---|
| Disk 01 | Dell | HHD | 1 TB | 6 | Windows 10 | Gmail Outlook | Webmail | No | 3 months | Yes |
| Disk 02 | MAC | HHD | 500 GB | 4 | OSX | Gmail | Webmail | Yes | Daily | No |
| Disk 03 | MAC | SSD | 250 GB | 1 | OSX | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 04 | HP | HHD | 500 GB | 3 | Windows 8 | Gmail Outlook Yahoo | Webmail Client | No | Daily | Yes |
| Disk 05 | Drive only | SSD | 112 GB | 2.5 | Windows 7 | Gmail Hotmail | Webmail | No | Daily | Yes |
| Disk 06 | HP | HHD | 130 GB | 3.7 | Windows 7 | Gmail | Webmail | No | Weekly | No |
| Disk 07 | MAC | SSD | 120 GB | 2.3 | OSX | Gmail Outlook Hotmail | Webmail | No | Daily | Yes |
| Disk 08 | MAC | SSD | 120 GB | 1 | OSX | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 09 | Surface | SSD | 250 GB | 0.9 | Windows 10 | Gmail Outlook | Webmail | Yes | Daily | Yes |
| Disk 11 | MAC | HHD | 465 GB | 5 | OSX | Gmail Outlook Apple | Webmail Client | Yes | Daily | Yes |
| Disk 12 | Surface | SSD | 128 GB | 1 | Windows 10 | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 13 | HP | HHD | 465 GB | 4 | Windows 10 | Gmail Outlook | Webmail Client | Yes | Daily | Yes |
| Disk 14 | Toshiba Satellite | HHD | 750 GB | 2 | Windows 10 | Gmail Outlook Yahoo | Webmail | No | Daily | No |
| Disk 15 | Lenovo 580 | HHD | 1 TB | 4.2 | Windows 10 | Gmail Outlook | Webmail | No | Daily | No |
| Disk 16 | Dell | HHD | 1 TB | 2 | Windows 10 | Gmail Yahoo | Webmail | Yes | Weekly | No |
| Disk 17 | Acer | HHD | 220 GB | 4 | Windows 7 | Gmail Yahoo College | Webmail | No | Daily | No |
| Disk 18 | Mac | HHD | 500 GB | 4 | OSX | Gmail Outlook | Webmail | Yes | Weekly | No |
| Disk 20 | Mac | HHD | 500 GB | 2 | OSX | Gmail Outlook | Webmail | No | Daily | No |
| Disk 21 | Razer Blade | SSD | 477 GB | 1 | Windows 10 | Gmail Outlook Hotmail | Webmail Client | No | Daily | Yes |
| Disk 22 | HP Pavillion | HHD | 465 GB | 0.1 | Windows 10 | Outlook | Webmail | No | Monthly | No |
| Disk 23 | Surface | SSD | 240 GB | 1.6 | Windows 10 | Outlook Yahoo | Webmail | No | Daily | Yes |
| Disk 24 | MacPro | SSD | 128 GB | 2.7 | OSX | Gmail Outlook Hotmail Apple | Webmail Client | No | Daily | Yes |

## Table C.1

| Disk Name | Brand of Computer | Drive Type | Size of Drive | Age of Drive(yrs) | Operating System | Email Provider | Email Access Method | Shared Computer | Communication Frequency | VM Software |
|---|---|---|---|---|---|---|---|---|---|---|
| Disk 26 | Mac Pro | HHD | 500 GB | 8 | OSX | Gmail Outlook Yahoo | Webmail | Yes | Daily | Yes |
| Disk 27 | Hard drive | HHD | 1 TB | 5 | Windows 10 | Yahoo Hotmail Verizon | Webmail | Yes | Weekly | No |
| Disk 28 | ASUS | HHD | 932 GB | 0.4 | Windows 7 | Gmail Outlook | Webmail Client | Yes | Daily | Yes |
| Disk 30 | surface | SSD | 115 GB | 0.9 | Windows 10 | Gmail Outlook | Webmail | No | Daily | Yes |
| Disk 31 | MacAir | SSD | 115 GB | 1.7 | OSX | Gmail Outlook Apple | Webmail Client | No | Daily | Yes |
| Disk 32 | HP | SSD | 240 GB | 0.7 | Windows 10 | Gmail Outlook | Webmail | No | Daily | Yes |
| Disk 33 | Dell | HHD | 120 GB | 3 | Windows 10 | Gmail Outlook | Webmail | Yes | Daily | No |
| Disk 34 | Alienware | HHD | 930 GB | 3 | Windows 10 | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 35 | Alienware | SSD | 400 GB | 3 | Ubuntu 16.05 | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 36 | Lenovo 320 | HHD | 932 GB | 0.5 | Windows 10 | Outlook | Webmail Client | No | Daily | Yes |
| Disk 37 | Mac | HHD | 700 GB | 8 | OSX | Gmail Outlook Yahoo Apple | Webmail Client | No | Daily | Yes |
| Disk 38 | Dell | HHD | 930 GB | 2 | Windows 10 | Gmail Outlook Yahoo | Webmail | Yes | Monthly | No |
| Disk 39 | Surface | SSD | 240 GB | 1 | Windows 10 | Gmail Outlook | Webmail | No | Weekly | Yes |
| Disk 42 | Mac | SSD | 400 GB | 1.5 | OSX | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 43 | Mac | HHD | 466 GB | 5 | OSX | Gmail Yahoo Apple | Webmail Client | Yes | Daily | Yes |
| Disk 44 | Mac | SSD | 932 GB | 2 | OSX | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 45 | Dell | SSD | 1 TB | 0.9 | Windows 10 | Gmail Outlook | Webmail Client | No | Daily | Yes |
| Disk 46 | Razer | SSD | 940 GB | 0.3 | Windows 10 | Gmail Outlook | Webmail | No | Weekly | Yes |
| Disk 47 | Acer | HHD | 477 GB | 1 | Windows 10 | Gmail Outlook | Webmail | No | Daily | No |
| Disk 48 | Dell | HHD | 932 GB | 2.7 | Windows 10 | Gmail Outlook | Webmail Client | No | Daily | No |
| Disk 49 | MacPro | HHD | 500 GB | 7.4 | OSX | Gmail Outlook Yahoo | Webmail | No | Daily | No |
| Disk 50 | Asus | HHD | 300 GB | 5 | Windows 7 | Gmail | Webmail | No | Daily | Yes |
| Disk 51 | Lenovo | HHD | 932 GB | 0.9 | Windows 8 | Gmail Outlook | Webmail Client | Yes | Daily | No |

# APPENDIX D:
# "Useful" versus "Not Useful" Subgraphs

This table is showing useful subgraphs and not useful subgraphs on each participant drive. "Useful" subgraphs are identified with a "yes" and "not useful" subgraphs are identified with a "no".

Table D.1

| disk | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **disk01** | No | Yes | No | No | No | No | No | No | No | No |
| **disk02** | No | No | No | No | No | Yes | Yes | No | No | No |
| **disk03** | Yes | No | No | No | No | No | No | No | No | No |
| **disk04** | No | Yes | No | No | No | No | No | No | No | No |
| **disk05** | No | Yes | No | No | No | No | No | No | No | No |
| **disk06** | Yes | Yes | Yes | No | No | No | No | No | No | No |
| **disk07** | No | Yes | No | No | No | No | No | No | No | No |
| **disk08** | No | Yes | No | No | No | No | No | No | No | No |
| **disk09** | No | Yes | No | No | No | Yes | No | No | No | No |
| **disk11** | Yes | No | No | No | No | No | No | No | No | No |
| **disk12** | No | Yes | No | No | No | No | No | No | No | No |
| **disk13** | No | Yes | No | No | No | No | Not | No | No | No |
| **disk14** | No | Yes | No | Yes | Yes | No | No | No | No | No |
| **disk15** | Yes | Yes | No | Yes | Yes | Yes | No | No | No | No |
| **disk16** | No | No | Yes | No | No | No | No | No | No | No |
| **disk17** | No | No | Yes | No | No | No | No | No | Yes | No |
| **disk18** | Yes | No | No | No | No | No | No | No | No | No |
| **disk20** | Yes | No | No | No | No | No | No | No | No | No |
| **disk21** | No | Yes | No | No | No | No | No | No | No | No |
| **disk22** | No | No | No | No | No | No | No | No | No | Yes |
| **disk23** | No | Yes | No | No | No | No | No | No | No | No |
| **disk24** | Yes | No | No | No | No | No | No | No | No | No |

**Table D.1 continued from previous page**

| disk | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **disk26** | No | Yes | No | Yes | No | No | Yes | No | No | No |
| **disk27** | No | No | Yes | No | No | No | No | No | No | No |
| **disk28** | No | Yes | No | No | No | No | No | No | No | No |
| **disk30** | No | Yes | No | No | No | No | No | No | No | No |
| **disk31** | No | Yes | No | Yes | No | No | No | No | No | No |
| **disk32** | No | No | No | Yes | No | No | No | No | No | No |
| **disk33** | Yes | No | No | No | No | No | No | No | No | No |
| **disk34** | Yes | Yes | No | No | No | No | No | No | No | No |
| **disk35** | Yes | No | No | No | No | No | No | No | No | No |
| **disk36** | No | Yes | No | No | No | No | No | No | No | No |
| **disk37** | Yes | No | No | No | No | No | No | No | No | No |
| **disk38** | Yes | No | No | No | No | No | No | No | No | No |
| **disk39** | No | No | No | No | No | No | No | No | No | No |
| **disk42** | Yes | No | No | No | No | No | No | No | No | No |
| **disk43** | Yes | No | No | No | No | No | No | No | No | No |
| **disk44** | No | Yes | No | No | No | No | No | No | No | No |
| **disk45** | No | No | Yes | No | No | No | No | No | No | No |
| **disk46** | Yes | No | No | No | No | No | No | No | No | No |
| **disk47** | No | No | Yes | Yes | Yes | No | No | No | No | No |
| **disk48** | Yes | No | No | No | No | No | No | No | No | No |
| **disk49** | Yes | No | No | No | No | No | No | No | No | No |
| **disk50** | Yes | No | No | No | No | No | No | No | No | No |
| **disk51** | Yes | No | No | No | No | No | No | No | No | No |

# List of References


[1] D. Quick and K.-K. R. Choo, "Impacts of increasing volume of digital forensic data: A survey and future research challenges," *Digital Investigation*, vol. 11, no. 4, pp. 273–294, 2014.

[2] S. L. Garfinkel, "Digital media triage with bulk data analysis and bulk_extractor," *Computers & Security*, vol. 32, pp. 56–72, 2013.

[3] J. Green, "Constructing social networks from secondary storage with bulk analysis tools," M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA USA, 2016.

[4] G. Allen, "Constructing and classifying email networks from raw forensic images," M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, USA, 2016.

[5] J. Myers and M. Rose, "Post office protocol-version 3," May 1996, [Online]. Available: https://tools.ietf.org/html/rfc1939

[6] M. R. Crispin, "Internet message access protocol-version 4rev1," March 2003, [Online]. Available: https://tools.ietf.org/html/rfc1733

[7] "How to choose between APFS and Mac OS Extended when formatting a disk for Mac - Apple Support". Apple. [Online]. Available: https://support.apple.com/en-us/HT208033

[8] A. Al Mamun, G. Guo, and C. Bi, *Hard disk drive: mechatronics and control*. Boca Raton, Florida, USA: CRC press, 2006.

[9] L. Hutchinson, "Solid-state revolution: in-depth on how ssds really work," 2012, [Online]. Available: https://arstechnica.com/information-technology/2012/06/inside-the-ssd-revolution-how-solid-state-disks-really-work/

[10] "What image formats do accessdata products support? – accessdata help center," [Online]. Available: https://support.accessdata.com/hc/en-us/articles/222778608-What-Image-Formats-Do-AccessData-Products-Support

[11] J. R. Bradley and S. L. Garfinkel, "Bulk extractor 1.4 programmers manual for developing scanner plugs-ins," Naval Postgraduate School, Monterey, CA, USA, Tech. Rep., 2013, [Online]. Available: https://calhoun.nps.edu/handle/10945/36029

[12] "GEXF File Format". Gephi. [Online]. Available: https://gephi.org/gexf/format/

[13] M. McCarrin, J. Green, and R. Gera, “Constructing social networks for digital forensic investigations,” presented at NetSci Conference, 2017.

[14] A.-L. Barabási, *Network Science*. Cambridge, United Kingdom: Cambridge University Press, 2016.

[15] L. Euler, “Leonhard Euler and the Königsberg Bridges,” *Scientific American*, vol. 189, no. 1, pp. 66–70, 1953.

[16] B. Bollobás, *Modern Graph Theory*. New York, New York, USA: Springer Science & Business Media, 1998.

[17] G. Chartrand and P. Zhang, *A first course in graph theory*. Mineola, New York, USA: Courier Corporation, 2012.

[18] M. E. Newman, “The mathematics of networks,” *The New Palgrave Encyclopedia of Economics*, vol. 2, no. 2008, pp. 1–12, 2008, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.8175&rep=rep1&type=pdf

[19] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008, [Online]. Available: http://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/meta

[20] S. B. Seidman, “Network structure and minimum degree,” *Social Networks*, vol. 5, pp. 269–287, 1983, [Online]. Available: https://www.sciencedirect.com/science/article/pii/037887338390028X

[21] “Features,” Gephi, 2018, [Online]. Available: https://gephi.org/features/

[22] M. Bastian, S. Heymann, M. Jacomy *et al.*, “Gephi: an open source software for exploring and manipulating networks.” *ICWSM*, vol. 8, pp. 361–362, 2009.

[23] D. M. Powers, “Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation,” *Journal of Machine Learning Technologies*, vol. 2, no. 1, 12 2001, [Online]. Available: http://hdl.handle.net/2328/27165

[24] B. Carrier, “Open source digital forensics tools: The legal argument,” Stake, Tech. Rep., 2002, [Online]. Available: https://packetstorm.foofus.com/papers/IDS/atstake_opensource_forensics.pdf

[25] M. D. Kohn, M. M. Eloff, and J. H. Eloff, “Integrated digital forensic process model,” *Computers & Security*, vol. 38, pp. 103–115, 2013, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404813000849

[26] M. Reith, C. Carr, and G. Gunsch, “An examination of digital forensic models,” *International Journal of Digital Evidence*, vol. 1, no. 3, 2002, [Online]. Available: http://www.just.edu.jo/~Tawalbeh/nyit/incs712/digital_forensic.pdf

[27] V. Baryamureeba and F. Tushabe, “The enhanced digital investigation process model,” in *Proceedings of the Fourth Digital Forensic Research Workshop*, 2004, pp. 1–9, [Online]. Available: https://dfrws.org/sites/default/files/session-files/paper-the_enhanced_digital_investigation_process_model.pdf

[28] “Electronic crime scene investigation: A guide for first responders, second edition,” April 2008, [Online]. Available: https://www.ncjrs.gov/pdffiles1/nij/219941.pdf

[29] F. Freiling and B. Schwittay, “A common process model for incident response and digital forensics,” *Proceedings of the IMF2007*, 2007, [Online]. Available: https://www.imf-conference.org/imf2007/2%20Freiling%20common_model.pdf

[30] A. Agarwal, M. Gupta, S. Gupta, and S. C. Gupta, “Systematic digital forensic investigation model,” *International Journal of Computer Science and Security (IJCSS)*, vol. 5, no. 1, pp. 118–131, 2011, [Online]. Available: https://www.researchgate.net/profile/Yatendra_Gupta/publication/228410430_Systematic_Digital_Forensic_Investigation_Model/links/56ea8cd208ae95bddc2bcc6b/Systematic-Digital-Forensic-Investigation-Model.pdf

[31] T. J. Andrzejewski, “Scaling bulk data analysis with mapreduce,” Ph.D. dissertation, Monterey, California, USA: Naval Postgraduate School, 2017, [Online]. Available: https://calhoun.nps.edu/handle/10945/56132

[32] S. L. Garfinkel, “Digital forensics research: The next 10 years,” *Digital Investigation*, vol. 7, pp. S64–S73, 2010, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287610000368

[33] B. Carrier *et al.*, “Defining digital forensic examination and analysis tools using abstraction layers,” *International Journal of digital evidence*, vol. 1, no. 4, pp. 1–12, 2003, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9813&rep=rep1&type=pdf

[34] E. Casey, *Digital evidence and computer crime: Forensic science, computers, and the internet*. Cambridge, Massachusetts, USA: Academic press, 2011.

[35] L. Garber, “Encase: A case study in computer-forensic technology,” *IEEE Computer Magazine January*, 2001, [Online]. Available: http://www.cosgrovecomputer.com/documents/computer_magazine_article.pdf

[36] D. Manson, A. Carlin, S. Ramos, A. Gyger, M. Kaufman, and J. Treichelt, "Is the open way a better way? digital forensics using open source tools," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*. IEEE, 2007, pp. 266b–266b, [Online]. Available: https://ieeexplore.ieee.org/document/4076922/#full-text-section

[37] N. C. Rowe, "Identifying forensically uninteresting files using a large corpus," in *International Conference on Digital Forensics and Cyber Crime*. New York City, NY: Springer, 2013, pp. 86–101, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-14289-0_7

[38] N. C. Rowe, R. Schwamm, M. R. McCarrin, and R. Gera, "Making sense of email addresses on drives," *The Journal of Digital Forensics, Security and Law: JDFSL*, vol. 11, no. 2, p. 153, 2016, [Online]. Available: https://search.proquest.com/docview/1825887617?pq-origsite=gscholar

[39] V. K. Devendran, H. Shahriar, and V. Clincy, "A comparative study of email forensic tools," *Journal of Information Security*, vol. 6, no. 2, p. 111, 2015, [Online]. Available: https://digitalcommons.kennesaw.edu/facpubs/3612/

[40] S. Boucher and B. Kuange, "Email evidence-now you see it, now you don't," [Online]. Available: http://www.forensicfocus.com/email-evidence-now-you-see-it

[41] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 137–143, [Online]. Available: https://dl.acm.org/citation.cfm?id=1138016

[42] R. F. Marineau, "The birth and development of sociometry: The work and legacy of jacob moreno (1889–1974)," *Social Psychology Quarterly*, vol. 70, no. 4, pp. 322–325, 2007, [Online]. Available: https://doi.org/10.1177/019027250707000402

[43] L. Freeman, "The development of social network analysis," *A Study in the Sociology of Science*, vol. 1, 2004, [Online]. Available: https://www.researchgate.net/profile/Linton_Freeman/publication/239228599_The_Development_of_Social_Network_Analysis/links/54415c650cf2e6f0c0f616a8/The-Development-of-Social-Network-Analysis.pdf

[44] J. Diesner, T. L. Frantz, and K. M. Carley, "Communication networks from the enron email corpus "it's always about the people. enron is no different"," *Computational & Mathematical Organization Theory*, vol. 11, no. 3, pp. 201–228, 2005, [Online]. Available: https://link.springer.com/article/10.1007/s10588-005-5377-0

[45] A. Chapanond, M. S. Krishnamoorthy, and B. Yener, “Graph theoretic and spectral analysis of enron email data,” *Computational & Mathematical Organization Theory*, vol. 11, no. 3, pp. 265–281, 2005, [Online]. Available: https://link.springer.com/article/10.1007/s10588-005-5381-4

[46] S. Bonner, J. Brennan, G. Theodoropoulos, I. Kureshi, and A. McGough, “Efficient comparison of massive graphs through the use of ’graph fingerprints’.” in *Twelfth Workshop on Mining and Learning with Graphs (MLG) at KDD’16*. ACM, August 2016, [Online]. Available: http://dro.dur.ac.uk/19773/

[47] S. Liu and I. Lee, “A hybrid sentiment analysis framework for large email data,” in *Intelligent Systems and Knowledge Engineering (ISKE), 2015 10th International Conference on*. IEEE, 2015, pp. 324–330, [Online]. Available: https://ieeexplore.ieee.org/document/7383067/#full-text-section

[48] “Gephi - the open graph viz platform,” Gephi, [Online]. Available: https://gephi.org

[49] T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129,1164, 199111, [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380211102

[50] E. de Castro Lopo, “Secret rabbit code (aka libsamplerate),” [Online]. Available: http://www.mega-nerd.com/libsamplerate/

[51] E. de Castro Lopo, “Libsndfile–a c library for reading and writing sound files,” [Online]. Available: http://www.mega-nerd.com/libsndfile/FAQ.htmlQ

[52] “SO File Extension - What is a .so file and how do I open it?” [Online]. Available: https://fileinfo.com/extension/so

[53] “How Do I Populate The AutoComplete Name List In A New Outlook Profile? - Intermedia Knowledge Base,” [Online]. Available: https://kb.intermedia.net/article/1894

THIS PAGE INTENTIONALLY LEFT BLANK

# Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California