Forensic Multimedia File Carving

Master’s Thesis

Author: Muhammad Nadeem Ashraf

Department of Computer and Systems Sciences

Royal Institute of Technology

Summer 2012

Supervisor (KTH): Prof. Dr. Oliver Popov

Supervisor (Fraunhofer SIT): Dr. Martin Steinebach
Abstract

Distribution of video contents over the Internet has increased drastically over the past few years. With technological advancements and emergence of social media services, video content sharing has grown exponentially. An increased number of cyber crimes today belong to possession or distribution of illegal video contents over the Internet. Therefore, it is crucial for forensic examiners to have the capability of recovering and analyzing illegal video contents from seized storage devices.

File carving is an advanced forensic technique used to recover deleted contents from a storage device even when there is no file system present. After recovering a deleted video file, its contents have to be analyzed manually in order to classify them. This is not only very stressful but also takes a large amount of time. In this thesis we propose a carving approach for streaming multimedia formats that allows forensic examiners to recover individual frames of a video file as images. The contents of these images then can be classified using existing techniques for forensic analysis of image sets.

A carving tool based on this approach is developed for MPEG-1 video files. A number of experiments are conducted to evaluate performance of the tool. For each experiment an MPEG-1 file with different encoding parameters is used. Moreover, each experiment contains 18 runs and with each run chunk size of the input MPEG-1 file is varied in order to create different amount of disk fragmentation.

For video only MPEG-1 files, 87.802 % frames are fully recovered when the chunk size is equal to 124 KB. Whereas in the case of MPEG-1 files containing both audio and video data 90.55 % frames are fully recovered when the chunk size is 132 KB.

**Keywords:** Data recovery, File carving, File fragmentation, Streaming multimedia
I dedicate this work to my loving parents...
I would like to express my deepest gratitude to Dr. Martin Steinebach for giving me the opportunity to work at Fraunhofer Institute. His continuous guidance and critical feedback helped me staying in the right direction to achieve goals of the thesis. I would also like to thank my supervisor at Royal Institute of Technology, Professor Dr. Oliver B. Popov for his valuable feedback on the thesis.

Especially I would like to thank Mr. York Yannikos and Mr. Christian Winter from Fraunhofer Institute for their continuous help throughout the course of the thesis work.

Last but not least I would like to thank my parents. Without their prayers and support it was not possible to achieve anything in my life.
Table of Contents

List of Abbreviations ........................................................................................................ VI
List of Figures .................................................................................................................. VII
List of Tables..................................................................................................................... VII

1 Introduction ......................................................................................................................
   1.1 Background ............................................................................................................... 1
   1.2 Problem Statement .................................................................................................... 1
   1.3 Goal and Purpose ...................................................................................................... 2
   1.4 Limitations and Scope .............................................................................................. 2
   1.5 Method ...................................................................................................................... 2
   1.6 Thesis Organization ................................................................................................. 3

2 Data recovery and file carving .........................................................................................
   2.1 Data Recovery ......................................................................................................... 4
   2.2 File Carving ............................................................................................................. 5
      2.2.1 Magic Numbers ................................................................................................. 6
      2.2.2 File Carving Techniques ................................................................................... 6
   2.3 File Carving VS Stream Carving .............................................................................. 7
      2.3.1 Example of Stream Carving ............................................................................ 8

3 MPEG-1 format ..............................................................................................................
   3.1 MPEG-1 File Structure ........................................................................................... 9
      3.1.1 Structure of MPEG-1 video ........................................................................... 11

4 Approach ....................................................................................................................... 15
   4.1 Carving Challenges ................................................................................................. 15
      4.1.1 Frame Size ....................................................................................................... 15
      4.1.2 Differences in Frame Structure ....................................................................... 15
      4.1.3 Frame Resolution ......................................................................................... 17
   4.2 Carving Algorithm ................................................................................................... 18
      4.2.1 Handling Frame Resolution .......................................................................... 22
      4.2.2 Fragmentation Handling .............................................................................. 24

5 Design .......................................................................................................................... 27
   5.1 System Overview .................................................................................................... 27
      5.1.1 Preprocessing ................................................................................................. 27
      5.1.2 Carver ............................................................................................................ 28
      5.1.3 Demultiplexer ............................................................................................... 28
      5.1.4 Decoder .......................................................................................................... 28

6 Experiments and Results ............................................................................................. 29

7 Conclusion and Future Work ....................................................................................... 43
7.1 Concluding Remarks ................................................................. 43
7.2 Future Work ........................................................................... 44

References .................................................................................. 45

Appendix A .................................................................................. 47
Bash script to store MPEG-1 file on raw data set.............................. 47

Appendix B .................................................................................. 48
Definition of Terms ...................................................................... 48
List of Abbreviations

3G  3rd generation mobile telecommunications
4G  4th generation mobile telecommunications
CD  Compact Disk
DFRWS  Digital Forensics Research Conference
ES  Elementary Stream
Ext3  Third Extended File System
FAT  File Allocation Table
GOP  Group of Pictures
GUI  Graphical User Interface
HFS  Hierarchical File System
ID3  Identify an MP3
JPEG  Joint Photographic Experts Group
MPEG  Moving Picture Experts Group
MP3  MPEG-1 Audio Layer 3
MP4  MPEG-4 Part 14
NTFS  New Technology File System
P2P  Peer to Peer
PES  Packetized Elementary Stream
SIF  Source input format
VCD  Video CD
VBV  Video buffering verifier
List of Figures

Figure 1: Metadata entry of a deleted file still pointing to file’s contents [4] ........................................ 5
Figure 2: MPEG-1 system stream [33] .................................................................................................. 10
Figure 3: MPEG-1 video stream [33] .................................................................................................. 12
Figure 4: I frame structure in video elementary stream ........................................................................ 16
Figure 5: I frame structure in system stream ......................................................................................... 17
Figure 6: GOP structure with two I frames .......................................................................................... 22
Figure 7: Decoding results of a frame with unique resolution values .................................................. 24
Figure 8: I frame with two fragments separated with random data ...................................................... 25
Figure 9: Carving results of a partial I frame ...................................................................................... 26
Figure 10: Components of the carving tool ......................................................................................... 27
Figure 11: Chunk size and number of fully recovered I frames in first data set ................................... 31
Figure 12: Example of less number of recovered I frames while increasing chunk size ...................... 31
Figure 13: Chunk size and number of fully recovered I frames in first five data sets .......................... 36
Figure 14: Chunk size and number of fully recovered I frames in last five data sets ........................... 42

List of Tables

Table 1: MPEG-1 start codes [36] ......................................................................................................... 10
Table 2: Picture coding type [37] ......................................................................................................... 14
Table 3: Carving results of first data set .............................................................................................. 30
Table 4: Carving results of second data set .......................................................................................... 32
Table 5: Carving results of third data set ............................................................................................. 33
Table 6: Carving results for fourth data set .......................................................................................... 34
Table 7: Carving results for fifth data set ............................................................................................. 35
Table 8: Carving results for sixth data set ........................................................................................... 37
Table 9: Carving results for Seventh data set ..................................................................................... 38
Table 10: Carving results for eighth data set ....................................................................................... 39
Table 11: Carving results for ninth data set ......................................................................................... 40
Table 12: Carving results for tenth data set ......................................................................................... 41
1 Introduction

1.1 Background

Video content sharing over the Internet has become highly popular in last few years. In the past, when internet was not developed enough in terms of speed and bandwidth, the users were required to download an entire multimedia file and then save the file on local disk before viewing its contents [46]. Today with the availability of very fast and affordable (in most countries) Internet connections, multimedia contents can now be streamed over the internet without the need to download them. In addition, a significant amount of video content is being distributed over Peer-to-Peer (P2P) file sharing techniques[47]. With the advent of powerful mobile computing devices (e.g. smart phones and tablet pcs) and high speed mobile data networks (e.g. 3g, 4g), together with social media services, the amount of video distribution over the Internet is growing exponentially.

In accordance with this, the amount of shared illegal videos has also increased. As in most countries not only the distribution but even the possession of certain image and video data is illegal, e.g. material containing child pornography, findings of such data during forensic investigations can be very important evidence. Therefore, the capability of recovering and analyzing video data is crucial for forensic examiners.

During digital forensic investigations, the investigators often encounter with a situation where they are required to recover deleted data from a seized storage device. Traditional data recovery techniques are based on file system information. Such techniques use file system metadata information to recover deleted files. In case when metadata information is not available or the file system itself is damaged, these techniques cannot be used to recover deleted files. That is the reason, we use advanced forensic techniques such as file carving that works completely independent of the underlying file system [5]. With file carving, deleted contents can be recovered as long as they are not over written.

Streaming multimedia formats are very suitable for carving due to their inherent file structure. These formats are rather stream of data that follow a different file structure than non multimedia formats. The actual audio and video contents of the file are divided into many number of small independent chunks called frames. Each frame starts with its own header which is followed by the frame contents. Here we do not rely only on the header and footer of the file but have much more information that can be used for carving. This makes it possible to start carving at any point in the file and extract individual frames when others are already overwritten.

1.2 Problem Statement

After forensic recovery of deleted video files, each file has to be manually screened in order to find presence of any illegal material. A video file is actually a combination of several still images which are called frames. A few minutes of video might consists of hundreds of these frames. Manually
screening each individual frame is not only a very stressful process but also takes a large amount of time.

While there is no reliable method for automatic screening of video files, there are several approaches we can use in case of image files. One such approach is discussed in [48] where robust hashing is used for the forensic analysis of image sets. We can apply the same existing techniques of image classification on video files if we can recover individual frames of a deleted video file instead of targeting the whole file.

1.3 Goal and Purpose

The goal of the thesis is to develop a carving technique for streaming multimedia formats by taking advantage of their file structure. This is a proof of concept evaluation in order to see if we can extract individual frames of streaming video files. Ultimately we want to develop a tool that carves individual video frames of deleted multimedia files so that presence of illegal material can be detected in these frames by applying image classification techniques. This will automate the whole process of recovering and screening deleted multimedia files during forensic investigations.

1.4 Limitations and Scope

This thesis is part of a research project carried out at Fraunhofer Institute for Secure Information Technology, Darmstadt Germany. The overall project aims to combine file carving techniques with techniques for forensic classification of image sets to automate the process of recovering and analyzing streaming video files. The scope of the thesis is limited to the first part only where the objective is to develop a carving technique for streaming video formats in order to extract their individual frames.

For the purpose of our research we selected Moving Picture Experts Group (MPEG-1) format since it is supported by most of video players, and free decoders are easily available. An MPEG-1 video normally contains three types of frames that are I, B, and P frames. We will focus on extraction of I frames only due to complexity of the MPEG-1 format. This has been discussed in detail in chapter 4. However, in this report we present detailed information of MPEG-1 file structure and carving algorithm for MPEG-1 I frames. This information can be used to explore the possibility of individually carving B and P types of frames in future.

1.5 Method

The thesis type is artifact development as our ultimate goal is to develop a carving algorithm and then a prototype using that algorithm which solves the stated problem.

The research is carried out mainly using the quantitative approach however qualitative approach has also been used partially. The research starts with the literature study in order to understand key concepts related to file carving and its techniques. File carving techniques can be divided into three main areas which are header-footer based, file structure based, and content based carving. The next phase involves study of the MPEG-1 file format and finding out suitable carving technique. The study
of MPEG-1 format shows that streaming multimedia formats have a different structure than other file formats. Here we have much more information of the file structure to use during the carving process. This motivated us to introduce a new carving technique called Stream Carving for streaming multimedia formats.

The prototype based on this approach has been developed in Java language. Java is easy to use and has a very effective mechanism for memory management and error handling. Several Java libraries have been used to simplify the programming task.

The performance of the prototype has been evaluated by conducting a series of experiments. The desired experimental results are in numerical form therefore qualitative method is used which is appropriate for this type of research. The experiments are carefully designed and simulate a real scenario with high disk fragmentation. After each experiment we count the number of fully recovered, partially recovered, and non-recovered I frames. The resultant number of frames are counted manually which involves human factor therefore the research method is also partially qualitative.

For each experiment, the percentage of fully recovered I frames is calculated. These figures are then used to generate graphs which can be used by a forensic investigator to estimate how much data can be recovered under a certain amount of disk fragmentation. The data collected through experiments is analyzed and the results are discussed to draw final conclusions.

1.6 Thesis Organization

The next chapter provides background information on data recovery and file carving. It describes different techniques of file carving and compares file carving with stream carving. In chapter 3 we discuss file structure of MPEG-1 streams that is necessary to understand in order to develop a carving approach for MPEG-1 files. Chapter 4 is divided into two sections. In the first section we discuss challenges of carving MPEG-1 files due to complex nature of the format. In the second section we present our approach and describe the carving algorithm. Chapter 5 presents design details of the carving tool implemented using the algorithm discussed in chapter 4. Next in chapter 6 we discuss experimental results that evaluates effectiveness of our approach. The conclusions, potential improvements in carving algorithm, and future work are part of the chapter 7.
2 Data recovery and file carving

2.1 Data Recovery

Data recovery is the process to restore damaged, failed or corrupted data from digital storage devices such as disks, tapes, and Compact Disks (CDs) [1] [2]. In context of digital forensics, data recovery techniques are used to restore deleted, damaged, or hidden data in a controlled environment. A forensically controlled environment is the one where examination is conducted by trained examiners and all actions are taken with their permission [3].

Traditional techniques to recover data depend on file system of the underlying operating system. A file system is a hierarchical structure where information about each file and its associated data is present. [4]. Common examples of file systems are File Allocation Table (FAT), New Technology File System for Windows, Third Extended File System (Ext3) for Linux, and Hierarchical File System (HFS) for MAC Operating system.

Each file system has a unique architecture; however, in general every file system contains information of file names, their associated metadata, and contents. These can be considered as three sections in the file system that are linked with each other. File name section stores an entry for each file that contains name of the file and address of its metadata entry. Metadata contains information about the file itself such as size, creation date and time etc. Metadata entries further points to the content section that holds information required to locate contents of the file on the physical disk. The content section stores information of all storage units of the physical drive. It keeps track of which storage units are allocated to which files and which one are unallocated. A storage disk is divided into fix size containers (storage units) called sectors [4]. Each file system groups these sectors which are commonly called blocks or clusters. A block or a cluster is the smallest portion of memory that can be addressed uniquely [5]. In this report we refer to disk storage units as blocks.

When creating a new file or adding data to an existing one, new blocks are assigned to the file. The state of a block is changed to unallocated in the file system when file is deleted, while the actual contents of a block are not removed from the disk. The only thing that happens is the file system marks the block as deleted and available for other files. Moreover, it depends on the file system whether or not it deletes the file name and metadata entries or keeps them. For example in FAT 32 when a file is deleted, the file system marks the file name entry as deleted by adding a special character to the file name and then it assigns the value zero to all blocks associated with the file [4][5].

Traditional data recovery techniques use metadata information in the file system to restore files. As metadata is linked with the content section that contains information of blocks assigned to a specific file. The recovery process in this case is just like reading undeleted files [4]. Figure 1 shows an example where metadata entry of a deleted file is still present and points to the blocks assigned to this file.
There are number of problems with this technique. Firstly it is up to the operating system to keep or delete metadata entries of a deleted file. Metadata entries of deleted files can also be reallocated to new files. In this case we lose link to the blocks assigned to the deleted file. Another issue is metadata can also get corrupted or can be lost for example if disk is formatted. It can also be manipulated manually by using wiping tools. In any such case, the traditional recovery techniques cannot be useful and we have to resort to more advance techniques to recover deleted data [5] [6].

2.2 File Carving

File carving is a technique that utilizes information of internal file structure and contents of a deleted file for recovery [5]. File carving does not depend on the file system and can recover files out of the raw data set. Traditional recovery techniques are relatively fast as we can see the only processing involved behind this is reading the file system. Carving is used mostly for files that are in unallocated space. This is the area that doesn’t have any metadata information referring it in the file system [5] [7].

An important concept in file carving is the handling of partial files also called fragmented files. There are different techniques implemented in operating systems to efficiently allocate blocks while creating a new file or adding data to an existing file. The operating system first searches for consecutive blocks but if not enough consecutive blocks are available then the file is stored on two or more locations. A file stored on multiple locations is called a fragmented file [4]. Common causes of fragmentation are low disk space and continuously appending more data to a file [5].

Traditional recovery techniques cannot recover a fragmented file if its metadata entry is not present that contains a link to blocks allocated to the file. With file carving it is possible to recover a file even if it is fragmented and stored on multiple locations in parts. This is because file carving techniques analyze a block or a set of blocks against characteristics of a specific file format and/or its contents [5].

Fragmentation handling is a difficult task and not all file carving techniques can handle fragmented files. Moreover, with current techniques it is not always possible to fully handle fragmentation and reconstruct the original file. In some cases it is required to recover all fragments of a file to make sense out of the recovered data and use it as evidence in the court. However, in certain cases it is
enough to recover partial file or parts of a file and use them as evidence. For example a video file consists of several still images. If such a video contains any offensive material like child pornography then legally it is not required to recover the whole file but if we can recover parts of the video file or even some of the individual images it is enough evidence to prove in the court.

The following section describes different file carving techniques and gives an idea of their effectiveness and the extent to which they handle fragmented files. Before going into further detail an important concept of magic numbers is required to understand first.

2.2.1 Magic Numbers

Most file formats use constant values called magic numbers or file signatures that is unique with that format and distinguishes it from other file formats. Files stored in a computer are associated with an extension such as .mp3, .doc etc. These extensions help the operating system to identify which application programs to use to open the file. Before opening a file, the application programs verify the signatures of the file to make sure this file has a correct format and is associated with the application [9]. These file signatures are normally used in a file to indicate start (Header) and end (Footer) of the file. As an example a Joint Photographic Experts Group (JPEG) file starts with the signature “0xFFD8” and ends with “0xFFD9”. [10]

2.2.2 File Carving Techniques

There are many file carving techniques that evolved gradually. We can categorize them first, second, and third generation carving techniques.

First generation

First generation carving tools such as Scalpel [13] used only file signature information to recover deleted files. This type of carving is referred to as header-footer based carving [11]. The idea is for a particular file, first search for the sequence of bytes consisting of the file’s header and then search for the first occurrence of the footer value in the same manner. Locations where header and footer values are found in the data set are marked and then all data blocks between these two locations are stored in an output file [8]. This is a very basic technique of file carving and it assumes that header and footer values of a file are not damaged or tempered with. Secondly it assumes the file is stored in contiguous data blocks means it is not fragmented [6] [9]. A slightly different technique to this is the header-maximum file size technique [11]. It is used for file types having only header information and no footer information. Header – maximum file size carving works in similar manner to the header – footer carving. First start of the file is located by searching its header. Then end of the file is calculated based on its size. For some file formats the size value is specified in the header of the file else we make an intelligent guess of the file size based on general observations of a specific file format. Once the start and end locations of a file are determined, data blocks between these locations are stored in an output file.

Header-footer and header-maximum file size based carving works only when deleted files are not fragmented and their signatures are not corrupted. Although this technique is very fast but it gives many false positives. The reason is if header and footer values are too short they can appear in the data set even when they do not actually represent a file [8]. As an example, JPEG header and footer values occupy only two bytes and there are strong chances of repetition of these values. This can
result in many such files in the output which are not actually JPEG files. For header – maximum size based carving, it is important to select an appropriate size if it is not specified in the header. Otherwise the recovered file may contain either too much irrelevant data appended at the end or might also lose relevant data [8]. Header – maximum size based carving was developed for formats such as JPEG that ignores any irrelevant data appended at the end [11].

**Second generation**

The second generation of file carvers used more information of the internal file structure to reduce file positives. This type of file carving is commonly called “File structure based carving”. It is also sometimes referred as “Semantic carving” or “Deep carving” [11]. File structure based carving works by first identifying certain level of information of a file format. This information is then matched in the raw data set to identify a file. File structure based carving reduces false positives greatly as more information of the file format is used. But just like header-footer or header-maximum file size based carving it cannot handle fragmented files [8].

**Third generation**

To handle file fragmentation more advanced techniques are used today. Main idea behind all these techniques is to read each individual block in the data set and analyze its contents to find out if it belongs to a particular file. These individual blocks are then rearranged if necessary to construct the original file. All such techniques which are based on reading individual blocks are called “Block content based carving” techniques [8] [11]. The difference in techniques lying under this category is in the method used for analyzing the contents of individual blocks.

Block content based carving works on two principles. First, a block can only belong to a single file [11]. Second, fragmentation can occur only at block boundaries [8]. The next step is to parse each block and apply calculations to determine which file this block belongs to. One such technique is to calculate entropy of each block. Blocks belonging to a certain file will have entropy value within a certain range [8] [14]. These blocks then have to be rearranged to construct original file. Smart carver from Digital Assembly [15] is one such tool that has implemented number of block content based and reassembly algorithms. Details of these algorithms can be found in Anandabrata Pal’s 2005 master thesis [16].

### 2.3 File Carving VS Stream Carving

A file usually starts with a header value which is followed by file contents and ends with the footer value. In addition to header value there could be more signatures in the file between header and content section.

A JPEG file for example is divided into three sections which are header, segments or markers, and trailer. JPEG header and footer values are 0xffd8 and 0xffd9 respectively. The middle section consists of different signature values called markers that contain information related to compression of image data. One of these markers is called SOS (start of scan). This is the point where actual image data starts. Image data is stored continuously and ends with the footer value which also marks end of the file [17]. Carving of such files is difficult in case the file is fragmented. For example a JPEG file is fragmented into two parts. The second part starts somewhere in the image content section, which
seems more likely to happen as size of all other markers is very small. Now it will be difficult to
detect the fragmentation point in this case and will require implementation of complex algorithms.

Some multimedia file formats such as MP3, MP4, and MPEG etc. are streaming formats and follow a
different file structure. Streaming formats may or may not have a beginning and end. The actual
contents are divided into small chunks where each chunk starts with its own header which is then
followed by the actual data. These chunks which are also called frames are independent units that can
be recognized by the relevant application. This type of format is very suitable for carving as for
identification of the file we do not rely only on the start and end signature values. Each frame is an
independent unit and has its own header which means we can randomly access sequence of blocks
starting with frame header and analyze it up to the end. One important point to mention here is that in
streaming media formats generally each frame contains some information which can be used to
calculate the frame length in bytes. This makes analyzing the sequence of frames very easy as we
know the location where next frame header should be present. Once we have found our first frame
and calculated its size, the next frame header should be present after X number of bytes where X is
the size of the previous frame. In case the frame header is not present at its expected location we can
assume end of the file is reached or the file is fragmented. In that case we can either leave the last
frame or carve it partially with other frames. We can then continue searching the data set for more
header values and if found can analyze the next frame or sequence of frames in the same manner.
Once we find all sequence of frames, the original file can be constructed back by ordering the
sequence of frames. This can be achieved by using knowledge of the internal file format. Even if
some frames are missing, partial files can be easily constructed.

2.3.1 Example of Stream Carving

Digital Forensics Research Conference (DFRWS) [18] forensic challenges 2006 and 2007 were based
on file carving. In 2007 challenge, Vassil Roussev developed a tool (MP3-scalpel) for MP3 file
carving which utilized only the characteristics of MP3 format for carving. An MP3 file consists of
independent audio frames where each frame starts with a header followed by CRC, Side information
and Main data. The actual audio data is stored in the Main data section. Each frame can be of different
length and length of the frame in bytes can be calculated with the following formula [12];

\[ \text{Frame length} = 144 \times \text{Bit Rate} / (\text{Sampling frequency} + \text{Padding}) \]

Information about the bit rate, sampling frequency and padding is present in each frame header.
Additionally, MP3 files also contain tags called ID3 (Identify and MP3) tags that hold meta-data of
the file [19]. There are two types of ID3 tags known as ID3v1 and ID3v2. ID3v1 tags are attached at
the end of the MP3 file whereas ID3v2 tags are at the start. An MP3 file may contain one or both of
these tags.

MP3-Scalpel first identifies all sequences of MP3 frames and ID3 tags present in the raw dataset.
These sequences and tags associated with a file are then combined to reconstruct the original file. To
identify a proper MP3 sequence, the carver starts with the header of the first frame, calculates its size
to obtain the position where next frame header should be present. All subsequent frames are verified
in this manner until the end is reached. The end of a sequence is identified if the next frame header is
not present at the expected position. For every sequence the carver searches for ID3v1 and ID3v2 tags
and considers them as start and end point of an MP3 file respectively [20].
3 MPEG-1 format

This chapter provides detailed description on MPEG-1 file structure. The carving algorithm we developed for MPEG-1 files is based only on its file structure information. Therefore it is necessary to understand MPEG-1 file structure before going into details of the carving algorithm which is explained in next chapter. We will first discuss general structure of MPEG-1 files and then structure of MPEG-1 videos. However we will not describe structure of MPEG-1 audio as the carving algorithm targets only on the video part.

MPEG was established in 1988. The first standard developed by the group named MPEG-1 (ISO/IEC-11172) was published in August 1993. The motivation behind development of MPEG-1 standard was to make it possible to store raw digital video and audio on CD. This is because optical discs can have only a single byte stream containing both audio and video data. Moreover optical discs operate at a constant speed whereas audio and video may have a variable rate [21] [32]. Raw digital video requires enormous amount of space for storage. On a CD drive, only 5 minutes of uncompressed video data can be stored without any audio data [21]. With MPEG-1, audio and video data is compressed with a bitrate up to around 1.5 Mbit/s. This allows to store 74 minutes of video with associated audio data to store on a 650 MB CD drive [22]. Moreover with this much compression it is also possible to transmit digital data over 1.554 and 2.048 Mbit/s channels [23]. Main applications of MPEG-1 are storing video on CDs (VCD), storing video as computer files and transmitting video data over the internet [24]. However MPEG-1 includes limited error protection mechanism which makes it unsuitable for transmission over unreliable mediums [25].

3.1 MPEG-1 File Structure

An MPEG-1 file may consist of only video data or both video and audio data. In the later case, audio and video data is multiplexed together to form a single stream. In addition to audio and video, a multiplexed MPEG-1 stream may also include optional user data. At first, audio and video data is separately encoded. The output audio and video from an encoder is called an Elementary Stream (ES). Elementary audio and video streams are encapsulated into packets along with optional user data. These packets are called PES packets and the resultant stream is called Packetized Elementary Stream (PES). In the last step, packetized elementary streams are multiplexed to form a single stream that is called the system stream [25]. System streams consists of at least one video stream and one or two audio streams [26].
Overall structure of the system stream is shown in figure 2. This is the structure of an MPEG-1 file stored on a CD or computer. The system stream has a layered architecture. At the top layer it is divided into so called Packs where each pack starts with a 32 bit start code 0x000001BA called pack_start_code. At the end of the last pack it contains a 32 bit code (0x000001B9) called ISO 11172_end_code which marks end of the stream. Some of these 32 bit codes used in MPEG-1 streams are listed in table 1. A pack consists of a pack header and a number of PES packets. Pack header is 12 bytes long and starts with the pack_start_code. PES packets inside a pack contain elementary stream
data and a pack can have data only from a single elementary stream. The first pack contains the system header that basically describes the system stream. Each system header contains information about all packetized elementary streams that are multiplexed together [33].

The packet layer, as shown in figure 2, represents PES packets that contain compressed audio or video stream data. A PES packet is divided into header and payload section. PES header starts with a 32 bit code ranging from 0x000001BC – 0x000001FF. The last byte in the code defines the stream id that tells which type of elementary stream data is present inside the packet [33]. For example, for video elementary stream data the value is in the range of 0xE0 – 0xEF and for audio data the value is 0xC0 – 0xFD [34]. This information is necessary to parse while decoding the system stream. The decoder reads the PES header, identifies the type of data inside the packet and then passes the payload to relevant decoder e.g. audio or video decoder [33].

In a PES packet both header and payload are of variable length. Structure of the PES header is given below [34];

- First 3 bytes contain PES start code value 0x000001.
- Next 1 byte defines the stream id.
- Next 2 bytes define the PES packet length. The value is equal to the total number of bytes immediately after these two bytes up to end of the PES packet. From this value we can calculate the size of the payload.
- After the length field, there can be an arbitrary number of padding bytes always set to the value 0xFF [33].
- Next 2 bytes contain optional extension data indicating decoder buffer size for the stream (elementary streaming inside the PES packet).
- Next 10 bytes are also for optional extension data. These bytes contain time stamps. There are two types of timestamps PTS (Presentation time stamp) and DTS (Decoder time stamp) each assigned 5 bytes. There can be either only PTS or both PTS and DTS present. As these time stamps are optional, when none of them are present then one byte will be added in the header having the fixed value 0x01.

After the optional fields immediately starts the payload section. As described above, the payload contains data from an elementary audio or video stream. MPEG-1 audio and video streams also have a layered structure. However these layers are not considered during packetization process and so the layers can break at boundaries of PES packets [35].

MPEG-1 files having only video data consist of a single video elementary stream. Such files are a subset of multiplexed MPEG-1 files. However structure of video only MPEG-1 files is very different as in this case there is only a single elementary stream which is not packetized and so there is no packet layer and pack layer in this case. The following section describes structure of an MPEG-1 video elementary stream.

3.1.1 Structure of MPEG-1 video

MPEG1 video streams are mainly a sequence of still images called frames. Each MPEG-1 video stream can be divided into several layers. The organization of different MPEG-1 video stream layers is given in Figure 3.
Sequence layer

The first layer in an MPEG-1 video is the sequence layer. A sequence is a collection of still images sharing some common properties such as resolution, bitrate etc. [28]. A sequence starts with a sequence header and ends with sequence end code 0x000001B7.

Sequence header is of variable length and starts with sequence start code 0x000001B3. Structure of the header is following [33];

- After the start code, the next 24 bits describe width and height of the images in the sequence. Both parameters are assigned 12 bits each.
- Next 4 bits describe pixel aspect ratio. It is followed by another four bits that tell frame rate of the sequence.
- 18 bits describe bit rate of the video.
- Next 1 bit is marker bit and is always set. Marker bit is followed by 10 bits of Video Buffering Verifier (VBV) buffer size that defines minimum decoder buffer size required to decode the sequence.
- Next 1 bit indicates if the video bit stream complies to constrained parameters or not.

* All Sequence Headers after the first in an MPEG Video Stream are optional.

** End of Macroblock only included if frame is a D-Picture.
Next 1bit if set is followed by 64 bytes of intra quantization matrix. It is followed by another bit which is if set indicates presence of 64 bytes non intra quantization matrix value.

The sequence header is followed by optional extension or user data having start codes 0x000001B5 and 0x000001B2 respectively. The extension data is reserved for the MPEG-2 format so it is actually not present in the MPEG-1 sequence [35]. The user data is of arbitrary number of bytes and is terminated when next start code occurs [33].

**Group of pictures layer**

The next layer is called Group of pictures (GOP) layer. Normally a GOP contains 15 or 18 pictures/frames [30]. The first frame in a GOP is always I frame and there should be at least one I frame present in the GOP. GOP is used to provide random access in a video sequence [24] [27]. There is at least one or arbitrary number of GOPs in a sequence.

GOP header starts with GOP start code 0x000001B8 and is 8 bytes long [36]. Structure of the GOP header is following [33];

- After the start code, next 25 bits tells time code of the first picture in GOP.
- Next 1 bit is a flag value that tells whether the GOP is open GOP or closed GOP. An open GOP refers to previous GOP to decode some pictures in the GOP, where as a closed GOP does not.
- Next 1 bit is also a flag value which is if set to ‘1’ indicates modification in the original sequence of frames in the GOP. Modification in the sequence may cause incorrect decoding of a frame.

As similar to the sequence header, GOP header may also contain extension or user data. GOP Extension is also reserved for MPEG-2 [35].

**Picture layer**

The primary unit of a video sequence is a picture or frame. Each GOP may contain four types of frames called I, B, P, and D frames. I and D frames are the only ones that can be decoded independently from the other frames in a GOP. However D frames are low quality frames used for fast searching in the video and are rarely used [39]. More details on these different types of frames can be found in ISO 1172-2.

A frame is divided into header and data. The length of a frame is different for I, B, and P frames. All three share some common parameters. Some additional parameters are present in B and P frame headers as well. The frame header starts with start code 0x00000100. Structure of the header is given below; [33]

- After the start code, the next 10 bits are for temporal reference value of a frame. Temporal reference refers to the display order of frames inside a GOP. The value starts from 0 for first frame to be displayed and increases by 1 for each subsequent frame.
- Next 3 bits describe the picture type. The values are given in table 2.
- Next 16 bits are for VBV delay. VBV delay works with VBV buffer size parameter as defined in the sequence header. VBV buffer size defines the size of the decoder buffer required to decode a sequence. When start code of an encoded picture is moved to the decoder buffer VBV delay specifies the delay before starting decoding of the whole picture [38]. More information can be found in [38] or ISO 1172-2.
For I frames, the next 1 bit is a flag value for extra information. If the bit is set to 1, one byte of extra information will be added. The process is repeated until extra information bit has a value of 0. In case of B and P frames some more information is added before extra information. However as we are only concerned here with I frame so we will not discuss that part here.

At the end of the picture header, extension or user data may occur similar to the sequence header and GOP header. The same rule applies here for the extension data; it is reserved for MPEG-2 [35].

<table>
<thead>
<tr>
<th>Code</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Forbidden</td>
</tr>
<tr>
<td>001</td>
<td>I frames</td>
</tr>
<tr>
<td>010</td>
<td>P frames</td>
</tr>
<tr>
<td>011</td>
<td>B frames</td>
</tr>
<tr>
<td>100</td>
<td>D frames</td>
</tr>
<tr>
<td>101</td>
<td>Reserved</td>
</tr>
<tr>
<td>110</td>
<td>Reserved</td>
</tr>
<tr>
<td>111</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

*Table 2: Picture coding type [37]*

### Slice, macroblock and block layers

The basic coding unit of a frame is called Block. Six blocks together make a Macroblock. In a macroblock there are four luminance (Y) blocks and two chrominance blocks (Cb, Cr). A frame is divided into slices that start with a slice header. A slice is just a combination of macroblocks that are encapsulated in slice header. In a frame there has to be at least one slice. There is no restriction on size of a slice in MPEG-1 video. A slice may contain at least one macroblock or it may contain all macroblocks of a frame. Slices are introduced for error handling purposes. Each slice in a frame is coded independent to other slices so that when a decoder parses video bit stream it can skip a slice and move to the next one [24] [27] [28].

Slice header is of variable length and starts with slice start code that ranges from 0x00000101 to 0x000001AF. The last byte of the slice start code are used as slice number that means there can be maximum 175 slices in a frame. Structure of the slice header is given below [33];

- After the start code, next 5 bits are used for quantizer scale factor.
- The remaining part of the header contains extra information in the same manner as defined in the frame header.
4 Approach

In section 2.3 we introduce the concept of stream carving and explained how it is suitable to carve multimedia streaming file formats by utilizing information of their internal file structure. We show that streaming multimedia file formats have a different structure where the file contents are divided into small chunks called frames. Each frame has its own header and each frame is an independent unit.

We use the same concept in our carving algorithm for MPEG-1 files. As described in previous chapter, MPEG-1 videos mainly consist of three types of frames that are I, B, and P frames. However the carving algorithm recovers only I frames. This is because after carving, a frame has to be decoded in order to view its contents and I frames are the only one that can be decoded independently. To decode P and B frames, we need to have other frames from the GOP. The process of decoding only an I frame is different from decoding of an MPEG-1 stream. This will be discussed in the next chapter.

There are some challenges in applying the concept of stream carving on MPEG-1 format due to complexity of its structure. The following section discusses these challenges as requirements and in the subsequent section we discuss our carving algorithm and how these requirements are met.

4.1 Carving Challenges

4.1.1 Frame Size

Unlike other multimedia formats MPEG-1 frame headers do not contain any information that can be used to calculate size of the frame. Moreover frame sizes are variable and frames do not have an end code or a footer value that marks end of the frame. However the scenario is slightly different in our case as we are not interested to parse the whole sequence of frames but only individual I frames which are the first frames in a group of pictures. But still it is important in order to avoid losing valid frame data or addition of irrelevant data at the end of the frame. So the first requirement is to carefully locate end point of a frame.

4.1.2 Differences in Frame Structure

The second and most important requirement is to handle different MPEG-1 file structures. An MPEG-1 file may consist of only a video elementary stream or a system stream. In both cases file structure is highly different and therefore I frames look differently. The two different structures are shown in figures 4 and 5 respectively.

Figure 4 shows the first scenario where I frame is part of a video only MPEG-1 file. This is the simplest form of an I frame. The frame starts with a frame header and is followed by arbitrary number of slices each having its own header. After the frame header and before the first slice there can be arbitrary number of bytes of optional user data. I frame ends where the next frame header starts which is a P frame as shown in figure 4. Carving I frame in this case requires to verify the frame header, presence of any optional data, and then verifying slices. A slice can be decoded independently from
other slices in the frame. To decode a frame successfully there should be at least one slice present. So we have to identify each slice by verifying its header.

![Diagram of I frame structure in video elementary stream]

Figure 4: I frame structure in video elementary stream

Figure 5 shows the second scenario where I frame is part of a system stream. It shows how audio and video data is encapsulated into packets and then packs. The I frame in this case is divided among several video packets as highlighted in the figure. The first video packet contains Sequence header, GOP header and I frame header. Slice data is divided into these three video packets. Some of the slice data comes immediately after the frame header which is in the first packet. The second packet also contains some slice data. The last packet contains the remaining slice data of the I frame, header of the next P frame and some slice data belonging to the P frame.

Like frames, a single slice may also span in more than one video packet. This is because a slice is of arbitrary size as already discussed in previous chapter. Additionally, during packetization process lower layers are not considered so we might also have slice headers at boundary of two PES packets. Figure 4 also shows that audio packets may come in between the video packets of a frame which further breaks continuity of a video frame.

To carve I frames in this scenario first requires extracting all slices belonging to the frame spanned across different packets and then concatenating the frame header and slices together. This will result in an I frame having structure similar to the one shown in figure 3. This extraction is necessary to decode the frame. There are actually three decoders that work together to decode an MPEG-1 system stream. At first the system stream is passed through a decoder that handles the system layer and extracts the video and audio elementary streams. The video and audio elementary streams are then passed to video and audio decoders respectively. So we have to separate system layer information and audio data and extract only the video elementary stream part of the frame to pass it to the video decoder.
We have to consider both types of I frame structures to develop carving strategy. There can be multiple MPEG-1 files present in the raw dataset and the files may belong to either of the structures. So the requirement is to carve I frames from multiple MPEG-1 files present in the raw data set that includes both types of MPEG-1 file i.e. video only MPEG-1 and multiplexed MPEG-1 files.

![I frame structure in system stream](image)

**4.1.3 Frame Resolution**

In order to visually present a frame we have to decode it and then convert it to an image format such as JPEG or Bitmap. To decode a frame correctly we need to read the frame resolution information and pass it to the decoder. The frame resolution information is present in the sequence header in two fields i.e. width and height as already described in previous chapter under sequence header structure. Now there is no information present in a frame header or somewhere else in the MPEG-1 structure that tells which sequence a frame is part of so that we may identify the relevant sequence header and read the frame resolution information.
4.2 Carving Algorithm

The first step is to locate all frames present in the raw data set. Frames are located by searching for the 4 byte frame start code 0x000000100 that marks start of the frame header. Next for each frame the frame type is verified to check if it is an I frame and then frame structure is verified in the subsequent bytes using only the information from internal MPEG-1 file structure as explained in the previous chapter. If the frame structure is verified successfully, the bytes consisting of the whole frame are carved and then are passed to the decoder. Once the frame is decoded it is converted into JPEG format and is then stored as a file. In previous section we describe two different structures of an I frame as shown in figures 4 and 5. This section explains how we handle both types of structures in our carving approach.

We will first begin with the I frame structure presented in figure 4. Before going into algorithm details one important point to mention here is as all headers in the MPEG-1 file are of variable length except that of GOP header, so we cannot validate size of these headers and calculate the location of next layer’s header. The algorithm just reads the subsequent bytes starting from the frame start code like reading data in the form of a stream. Carving algorithm for this type of structure works as following:

1. Read the frame header and identify if the frame type is I frame. Verify that the frame is present correctly.
2. If the frame type is I frame then read next bytes from the data set and verify the frame structure. If not then repeat step 1.
3. Read next 4 bytes and verify if it contains user data start code or slice start code. The user data is optional and is not required in the decoding process of the frame. So we just skip the next few bytes until we find 4 bytes of slice start code.
4. If slice start code is present, read slice header and verify its structure. Is slice start code is not present then go back to step 1. This is because as already that stated there should be at least one slice present in the frame to decode it successfully.
5. Read next bytes and verify all slices until next start code is frame start code of the next frame or GOP start code or sequence end code.
6. Read all bytes of the frame from start offset to end offset and pass them to the decoder.
7. Convert the decoded frame data to JPEG and write in an output file.

In this example we assume that all start codes and the frame are present intact in the raw data set and there is no fragmentation anywhere within the boundary of the frame. The following pseudocode gives an insight of the algorithm;

```java
for (i = 0; i < total number of frames found; i++) {
    Stream.skipBits(10); //Skip 10 bit of temporal reference value
    if (next 3 bits == 001) {
        Stream.skipBits(16); //Skip 16 bits of vbv delay
        while (next 1 bit == 1) {
            Stream.skipBytes(1); //Skip 1 byte of extra information
        }
        gotoNextStartCode(); //search next start code
        if (next start code == user data start code)
```

skipUserData(); //skip user data
gotoNextStartCode(); //search next start code

If(next start code is not in the range of slice start code)
  Repeat loop for next frame
else
  While(next start code is in the range of slice start code)
  {
    Stream.skipBits(5); // Skip next 5 bits of quantizer scale
    while (next 1 bit == 1)
    {
      Stream.skipBytes(1); //Skip 1 byte of extra information
    }
    gotoNextStartCode(); //search next start code
  }
If (next start code is frame start code, GOP start code, or sequence end code)
  {
    byte [] frame = Stream.read(startPosition, endPosition);
    //Decode the frame and write as a file
    writeToFile(filename, Decode(frame));
  }
}

For each frame the first 10 bits after the frame start code are of temporal reference which should be equal to 0 for the first frame in GOP. As I frame is the first frame in a GOP so we might want to validate the temporal reference value. There can be more than one I frame in a GOP and it is also possible that an MPEG-1 file consists only of I frames. This means we cannot rely on the temporal reference value and so we skip this and move forward. The next three bits of Picture coding type are used to check the frame type. If these bits are equal to 001 then the frame is an I frame. Next 16 bits of VBV delay again do not have a fixed value that we can validate so we skip them as well. The next part of frame header might contain extra information. This is validated by first reading the next one bit which is if set will be followed by one byte of extra information. This process goes on until the extra information bit is set to zero.

The frame header can be followed by optional user data which is verified by the user data start code 0x000001B2. Next the algorithm verifies every slice of the frame. Slices are identified by the presence of slice start code which should be in the range of 0x00000101 - 0x000001AF. Next to slice start code are 5 bits of quantizer scale which is an integer value in the range of 1 to 31. The value is 31 if all 5 bits are 1 so these bits are just ignored. Lastly slice header may include extra information if next 1 bit is set. This is similar to the extra information added in the frame header and is verified in the similar way. As we know slices are just a combination of macroblocks. Macroblocks do have a header but no start code values. So we just verify the frame structure up to slice layer which produces accurate results with no false positives which is basically the main objective of structure based file carving.

The algorithm mentioned above identifies all slices of the frame until the next bytes include frame start code (next frame after the current I frame), GOP start code, or sequence end code. If any one of
these start code is present then we consider end of the frame has reached. These start codes cover all possible scenarios. Recall figure 4, after the last slice in the I frame we have header of the next P frame. This is true when the current I frame is the first frame of the GOP or any frame (there can be more than one I frames in a GOP) except the last one. If the current I frame is the last I frame in the GOP then there are two possible next start code values which are;

- GOP start code: If after the end of the current GOP there is another GOP.
- Sequence end code: If the current GOP is the last GOP of the sequence.

Now the algorithm reads all data between start position (first byte of the frame start code) and end position (last byte before the next start code as explained above) of the frame. These are then sent to the decoder along with frame resolution. Once the frame is decoded it is converted into JPEG format and written as an output file.

Now let us consider the second scenario where the I frame structure is the one as shown in figure 5. The first four steps of the algorithm remain the same. Step number 5 and onwards would be as following;

5. Read next bytes containing slice data and verify structure of each slice until the next start code is the pack start code.
6. Read the next bytes, if PES audio start code is found then skip next bytes until again pack start code is found.
7. Read next bytes, if it contains PES video start code then read slice all slice data present in the video packet and verify their structure until pack start code is found.
8. Repeat step 6 and 7 until next start code is frame start code or GOP start code or sequence end code.
9. Read all bytes of the frame from start offset to end offset and pass them to the demultiplexer.
10. Pass demultiplexed frame data to decoder.
11. Convert the decoded frame data to JPEG and write it in an output file.

The following pseudocode gives an insight of the algorithm;

```plaintext
If(next start code is not in the range of slice start code)  
    Repeat loop for next frame
else
    While(next start code is in the range of slice start code)
    {
        Stream.skipBits(5);
        while (next 1 bit = 1)
        {
            Stream.skipBytes(1);
        }
        gotoNextStartCode();//search next start code
    }
If(next start code is frame start code, GOP start code, or sequence end code)
    Follow previous scenario
else if (next start code is pack start code, PES AUDIO CODE, or PES VIDEO CODE)
    while (true)```
{  
gotoNextStartCode();//search next start code
Switch (start code)
{
  case PACK_START_CODE:  
    break;
  case PES_AUDIO_CODE:  
    break;
  case PES_VIDEO_CODE:
    while(next start code is in the range of slice start code)
    {
      Stream.skipBits(5);
      while (next 1 bit = = 1 )
      {
        Stream.skipBytes(1);
      }
      gotoNextStartCode();
    }
    break;
  default:  
    break;
}
}
If (next start code is frame start code, GOP start code, or sequence end code)
{
  byte [] frame = Stream.read (startPosition, endPosition);
  byte [] demuxFrame = Demultiplex (frame);
  writeToFile(filename, Decode(demuxFrame));
}

As shown in figure 5 slice data is divided into three packets. Some slice data comes immediately after the I frame header in first video packet. The remaining slice data is in second and third video packets. There are also some audio packets between first and second video packets. Moreover, third video packet also contains frame header and some slice data of the next frame. In this case we have to first remove audio data and headers from pack and packet layers. Then we have to concatenate the frame header and slice data to construct a frame having structure similar to the one in previous scenario as shown in figure 4.

The algorithm first verifies the slice data that comes immediately after the frame header in similar way as explained before. If the next start code value is “pack start code” or “PES audio/video start code” then it means that I frame is part of the system stream else the first scenario applies here. We continue reading next bytes of data until we find start code value of next frame, GOP, or Sequence end. The presence of any of these start codes marks the end of the frame. Within this range of data, we ignore all occurrences of pack headers and PES audio packets as they are not part of I frame. For each PES video packet found, slice data inside the packet is verified. Now the algorithm reads all data between start and end position of the frame and passes it to the demultiplexer that removes audio packets, pack, and PES layer headers. The slice data is extracted from all video PES packets and is
then concatenated with the frame header. Now we have frame structure similar to the one in first scenario. The frame is then sent to the decoder with resolution information. Once decoded the frame is then converted into JPEG format and written as an output file.

4.2.1 Handling Frame Resolution

As explained above, the carved frame is sent to the decoder along with frame resolution information in order to decode the frame successfully. We also mentioned that there is no information present in the frame header or anywhere else in the MPEG-1 structure that links a frame to its sequence. Nevertheless, in some cases we can use information of the internal MPEG-1 file structure to identify the sequence header of some I frames. This depends on two factors. The first factor is the position of the I frame in a GOP and the second factor is the number of GOPs in a sequence.

![Figure 6: GOP structure with two I frames](image)

Let us consider the first factor. Normally a GOP contains only one I frame, but it is also possible to have more than one. Figure 6 shows a sequence with only one GOP. The GOP contains two I frames. Now it is possible to link the first I frame with its sequence header using some information of the MPEG-1 file structure. For each frame found in the raw data set we search the data set backwards within a certain threshold value to find the sequence header. If the sequence header is found in that range, we assume that the sequence header and the frame belong to each other. Then, we can read the resolution information from the sequence header and pass it to the decoder along with the frame.

The threshold value is calculated using the maximum size values of the sequence header and GOP header. The maximum size of the sequence header is 136 bytes and GOP header has a fixed size of 8 bytes. When the two values are added, we have in total 144 bytes. If we assume that there is no user data after the GOP header and sequence header then the 144 bytes gives us a maximum threshold value that is equal to number of bytes between start of sequence header and start of frame header. If user data is present either after sequence header or GOP header then this approach does not work as user data is of arbitrary size.

It is not very common in MPEG-1 files to have user data; hence this approach works well most of the time. For each frame found in the raw data set, the algorithm searches 144 bytes backwards from the
offset where frame header starts. If sequence header is found in these 144 bytes the algorithm assumes
the frame belongs to this sequence and reads the resolution information.

For the second I frame which is the last frame of GOP as shown in figure 6, this approach does not
work as there are number of other frames between first and last I frame each having arbitrary size.
This is because slices in a frame are of arbitrary size and we cannot predict maximum size of a slice.
Again this approach works well for most of the I frames based on the observation that usually a GOP
has only one I frame and that has to be the first frame of the GOP.

The second factor on which this approach depends is the number of GOPs in a sequence. A sequence
may contain only one GOP or arbitrary number of GOPs. In the first case this approach works well as
explained above. In the later case it works only for the first GOP that immediately comes after the
sequence header. For the rest of the GOPs in that sequence it is not possible to calculate a threshold
value. The reason is obvious as the distance of those GOPs from the sequence header increases
substantially and everything between them is of variable size.

For each frame where the sequence header could not be located, another approach is used to find out
the resolution information. In this case we search the raw data set and locate all sequence headers.
Next the resolution information from each sequence header is read and unique resolution values are
extracted. This is based on two observations of the MPEG-1 file format. First each sequence header
belonging to an MPEG-1 file has the same resolution information encoded inside. Secondly most of
the MPEG-1 files are encoded using Source input format (SIF) that uses three resolutions, i.e.
352x240, 352x288, or 320x240 to encode the videos. This means the total number of unique
resolution values is directly related to the number of different MPEG-1 files present in the raw data
set. This number is expected to be very small as most of the MPEG-1 files are encoded using SIF
resolutions.

After extracting the unique resolution values, each frame for which there is no associated sequence
header is decoded using all these unique resolution values. For this purpose, a graphical user interface
(GUI) is developed where such frames are displayed after decoding with all possible resolution values
one of which will be the correct value. The examiner then can choose the correctly decoded frame and
discard other copies of the frame as shown in figure 7.

Although this technique works well most of the times, some problems might occur with certain files.
For example, when there is only one MPEG-1 file present in the raw data set that contains only one
sequence header. It is possible that the sequence header gets corrupted or could be overwritten by
other data. In this case even if there are frames present in the data set, the carver will not be able to
output any frame, as it cannot decode those frames without having the frame resolution information.
4.2.2 Fragmentation Handling

One important issue to deal with in file carving is the fragmentation handling. In our approach we assume that file headers are not damaged and there is no fragmentation between the boundary of a frame in both types of frame structures shown in figure 4 and 5 respectively. Our main objective was to develop a carving solution for MPEG-1 files. Fragmentation within the frame’s boundary is handled up to a small extent only, which is due to very complex and variable nature of the MPEG-1 file format.

Fragmentation is difficult to handle and requires block content based carving approach where each block in the data set is analyzed to find out the file this particular block belongs to. This requires a number of other techniques in combination of the file structure information as already discussed in chapter 2. In addition, a rather difficult task in fragmentation handling is to find out the fragmentation point. We already discussed in chapter 2 how streaming formats can be helpful in finding out the fragmentation point by using information of the internal file structure but in MPEG-1 format we do not have this advantage because there is no information present in the frame header that can be used to calculate where we expect to start the next frame and so can identify fragmentation. Also a frame is further divided into other layers each having its own header again with no information to calculate the size of that layer’s payload. Another problem is different types of frame structures as shown in figure 4 and 5. Lastly fragmentation can be linear or nonlinear. Linear fragmentation is more common and is relatively easier to handle where as non-linear fragmentation handling is a much more difficult task.

The carving algorithm handles only linear fragmentation. The purpose is to carve partial frames in case fragmentation occurs inside a frame boundary. The objective is to get as much as available slice data in case of fragmentation. This is because as explained earlier there should be at least one slice present in a frame to decode successfully. As we know that slices are independent units of a frame and can be decoded irrespective of the other slices in the same frame. Even if a slice itself is present partially it still can be decoded.
The carving algorithm, when it reaches the slice layer keeps track of the last known complete slice. If for example it detects fragmentation somewhere within the next slice then it drops the next slice with all remaining frame data and carves the frame up to last known complete slice. To achieve this, the carving algorithm after deleting the first slice searches for the next slice header within a threshold value. If next slice header is found within that threshold it considers the previous slice to be present completely. The threshold value is set to 92 KB which is an average size of GOP plus average size of an I frame [40]. We selected this threshold value because a frame may consist of only one slice or arbitrary number of slices.

For example, let us consider an I frame with only one slice as shown in figure 8. A random file ‘X’ is inserted somewhere in the slice due to fragmentation. The remaining part of the slice and the frame continues where the file X ends.

<table>
<thead>
<tr>
<th>Frame header</th>
<th>Slice data</th>
<th>Random file data</th>
<th>Remaining slice data</th>
<th>Next start code (Marks end of the frame)</th>
</tr>
</thead>
</table>

*Figure 8: I frame with two fragments separated with random data*

Now there are two possible scenarios. First if size of X is greater than 92 KB then in this case the slice will be dropped completely and frame will not be carved. This is because the algorithm keeps track of the last known complete slice and there is only one slice in the frame. When the algorithm reaches the slice header it will continue reading next bytes and will expect next slice header or another start code that marks end of the frame within the threshold value. As size of X is greater than 92 KB so the algorithm will not find next start code within the threshold value and will not detect a complete slice.

The second scenario is when size of X is less than 92 KB e.g. X has a size of 10 KB. In this case the algorithm will be able to find next start code (marks end of the frame) within the threshold value. The slice will be considered as a complete slice even there is a fragmentation and random file data is present inside the slice. The frame will be carved and decoded in this case. The decoding results depend on where X is present in the slice.

Figure 9 shows different cases where random file data is added at different locations inside the slice. The I frame shown in figure 9 contains only one slice. However, same rules apply even when there are multiple slices in the frame.
Figure 9: Carving results of a partial I frame

a) Original image

b) Random data added in the middle

c) Random data added close to end

d) Random data added close to start
5 Design

This chapter provides design details of the carving tool developed using the approach discussed in previous chapter. The tool is developed in Java language. We selected java language due to many of its advantages. Java is easy to use, robust, and platform independent. It has an effective memory management and error handling mechanism [41]. Additionally, there are several libraries for java that facilitate the job of a programmer. GUI can also be developed fairly straightforward using a combination of available java editors and additional plugins.

5.1 System Overview

Following are the key components of the carving tool as shown in figure 10.

- Preprocessing
- Carver
- Demultiplexer
- Decoder

![Figure 10: Components of the carving tool](image)

5.1.1 Preprocessing

Each java application is allocated with a specific amount of memory in a heap. This is where objects are being stored. The carving application reads and process raw disk image as a byte array. Raw disk images can be of variable sizes. It is not possible to read and process a large sized disk image in a single run due to memory limitations. In preprocessing phase, size of the disk image is determined. If size is greater than 100 MB, the image is divided into chunks each having a maximum size of 100 MB.
5.1.2 Carver

This is the component where the carving algorithm is implemented. Individual chunks of the disk image are processed here one by one. For each chunk, the carver performs the following tasks;

- Identify frame headers present in the chunk. We used Boyer-Moore string search algorithm to locate headers in the raw dataset. Boyer-Moore is a fast searching algorithm that significantly reduces the time to search a particular value in a target string [42]. For our application we needed a modified implementation of the Boyer-Moore algorithm that searches a byte value (start code) in the target chunk that is a byte array. The java implementation of the byte search Boyer-Moore algorithm can be found in [43].
- Process each frame from the carving algorithm.
- Search for all sequence headers present in the chunk, read frame resolution information from each header and calculate the unique resolution values to be used for frames that cannot be associated with a sequence header.
- Pass the carved frame to the demultiplexer if it is a part of the system stream. The demultiplexer returns the extracted frame to the carver.
- Pass the carved frame along with resolution to the decoder.

5.1.3 Demultiplexer

The role of the demultiplexer is to extract video elementary data of the frame from the system stream frame structure. Demultiplexer performs the following tasks;

- Separate video packets from any audio packets present in the frame structure.
- Process each video packet, remove PES headers and extract slice data from each packet.
- Concatenate frame header with slice data in original order.
- Return frame to the carver.

5.1.4 Decoder

Each carved frame is decoded for a visual presentation to the investigator. An open source MPEG-1 elementary stream decoder [44] is used in the application. The decoder is extensively modified according to the needs of our carving tool. The original decoder parses the whole elementary video stream starting from the sequence layer. It has the capability to decode all types of MPEG-1 video frames such as I, B, and P frames. The decoder is modified to take a video stream as an input that starts directly from the picture layer. It can decode only I frames, hence it does not have the functionality required to decode B and P frames.

After decoding each frame, the decoder performs two additional tasks which are;

- Convert decoded frame to JPEG format and write image to the disk.
- Create thumbnail of each image and send it to GUI. To create image thumbnails we used a java library named Thumbnailator that can be found in [45].
6 Experiments and Results

In this chapter, we have discussed the experiments made and the subsequent results, which are part of the tool verification and validation processes. The experiments help us to evaluate the effectiveness of the carving tool by determining if it meets all the requirements and challenges defined earlier. For this purpose, we simulate an environment close to the real scenarios.

In a real scenario, a file can be stored in a number of fragments present at different location in the data set. Moreover, the file might not be present in its entirety and part of the file can be overwritten by some other data. We evaluate the carving tool under these circumstances and see how many number of I frames are carved that are present either fully or partially.

We use 10 different data sets in our experiments each having a different MPEG-1 file. Each experiment contains 18 tests that takes quite some time to perform the tests and record the results. Due to this we limited the number of data sets to 10 only. Equal numbers of data sets are used for the elementary stream and system stream MPEG-1 files. The data sets do not contain any file system on them and are completely filled with random data before storing MPEG-1 files. Block size of data sets and MPEG-1 files is set to 4KB which is the most common block size used in current operating systems.

For each MPEG-1 file, we select a random entry point and divide the remaining file into a number of equal sized chunks. These chunks are then stored on the data set sequentially but at different locations resulting in a highly fragmented environment. For this purpose we use a small bash script that utilizes linux dd command to copy input MPEG-1 file chunks and store them on the raw data set. The script is listed in Appendix A. The process of fragmentation is however not completely random as the file chunks are stored sequentially on the disk. Secondly the location on disk where a chunk is stored is calculated simply by addition of a fixed value with the location where previous chunk was stored.

We perform a number of runs on each data set. With each run we change the size of the chunks and observe the number of correctly recovered I frames. The assumption is that the best results are obtained when the chunk size is equal to the average size of a GOP plus the average size of an I frame. A typical GOP has 12 frames with pattern IBBPBBPBBPBB. There is one I frame, eight B frames, and three P frames in this pattern. The average size of I, B, and P frames are 18 KB, 6 KB, and 2.5 KB respectively. This results in to average GOP size equal to 73.5 KB and size of GOP plus I frame equal to 91.5 KB.

For each data set we start with a chunk size equal to the block size of 4 KB. With each next run the chunk size is incremented by 8 KB and in the last run the chunk size is set to 140 KB. Hence, there are in total 18 runs performed on each data set. For each run the number of chunks is calculated as following:

\[
\text{Total number of blocks in the input file} = \frac{\text{File size (KB)}}{\text{Block size (KB)}}
\]

\[
\text{Number of blocks after the entry point} = \text{Total number of blocks in the input file} - \text{entry point}
\]

\[
\text{Chunk size} = \text{Block size} \times \text{count}
\]

\[
\text{Number of chunks} = \frac{(\text{Number of blocks after the entry point} \times \text{Block size})}{\text{chunk size}}
\]
The count value is adjusted with each run to get the required chunk size.

The first data set contains an elementary stream MPEG-1 file with the resolution 352*240, frame rate of 29.97 frames per second, and the bit rate equal to 1.62 Mbit/s. The original file size is 7058 KB and it has total 68 I frames. With an input block size of 4 KB there are total 1765 (round figure) Blocks in the file. For each run on the data set we chose a random entry point block 693 that results in 1073 remaining. For the first run total number of chunks are calculated as following:

\[
\text{Number of blocks after entry point} = 1765 - 692 = 1073
\]
\[
\text{Count} = 1
\]
\[
\text{Chunk size} = 4 \times 1 = 4
\]
\[
\text{Total number of chunks} = \frac{1073 \times 4}{4} = 1073
\]

For the first run there are total 1073 chunks or fragments of the MPEG-1 file stored at different locations on the data set. These 1073 chunks contain 41 I frames out of which 28 frames are recovered. All 28 frames are recovered with less than 50 % due to the fragmentation inside the frame. The remaining 13 frames are not recovered due to absence of at least one slice after the frame header required for decoding a frame. Table 3 shows the percentage of recovered frames for each of the runs.

<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered 100%</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1073</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>68.29</td>
<td>31.71</td>
</tr>
<tr>
<td>12</td>
<td>358</td>
<td>9.76</td>
<td>0</td>
<td>17.07</td>
<td>63.41</td>
<td>9.76</td>
</tr>
<tr>
<td>20</td>
<td>215</td>
<td>26.83</td>
<td>0</td>
<td>29.27</td>
<td>39.02</td>
<td>4.88</td>
</tr>
<tr>
<td>28</td>
<td>154</td>
<td>36.59</td>
<td>2.44</td>
<td>34.15</td>
<td>24.39</td>
<td>2.44</td>
</tr>
<tr>
<td>36</td>
<td>120</td>
<td>48.78</td>
<td>0</td>
<td>24.39</td>
<td>24.39</td>
<td>2.44</td>
</tr>
<tr>
<td>44</td>
<td>98</td>
<td>58.54</td>
<td>2.44</td>
<td>19.51</td>
<td>19.51</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>83</td>
<td>75.61</td>
<td>17.07</td>
<td>4.88</td>
<td>2.44</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>72</td>
<td>70.73</td>
<td>0</td>
<td>9.76</td>
<td>17.07</td>
<td>2.44</td>
</tr>
<tr>
<td>68</td>
<td>64</td>
<td>78.05</td>
<td>0</td>
<td>7.32</td>
<td>14.63</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>57</td>
<td>78.05</td>
<td>2.44</td>
<td>9.76</td>
<td>9.76</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>52</td>
<td>78.05</td>
<td>4.88</td>
<td>7.32</td>
<td>9.76</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>47</td>
<td>82.93</td>
<td>0</td>
<td>9.76</td>
<td>7.32</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>43</td>
<td>85.37</td>
<td>0</td>
<td>12.2</td>
<td>2.44</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>40</td>
<td>75.61</td>
<td>2.44</td>
<td>9.76</td>
<td>14.63</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>37</td>
<td>90.24</td>
<td>0</td>
<td>2.44</td>
<td>7.32</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>35</td>
<td>85.37</td>
<td>0</td>
<td>9.76</td>
<td>4.88</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>33</td>
<td>85.37</td>
<td>0</td>
<td>4.88</td>
<td>9.76</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>31</td>
<td>85.37</td>
<td>0</td>
<td>4.88</td>
<td>7.32</td>
<td>2.44</td>
</tr>
</tbody>
</table>

*Table 3: Carving results of first data set*
Figure 11 shows graph representation of carving results for the first data set. The graph shows the percentage of fully recovered I frames for a given chunk size. The number of fully recovered I frames within a given chunk size is directly related to the frame sizes. MPEG-1 uses different encoding techniques for I, B, and P frames that results in different sizes of these frames. I, B, and P frames have an average size of 18 KB, 6 KB, and 2.5 KB respectively. The size of an encoded I, B, and P frame also varies within the video stream.

During the experiments we divide the input MPEG-1 file into several chunks and store them at different locations in the data set. Also with each run we change the size of the input chunks. This may also result in missing some I frames even if the chunk size is increased. This scenario is shown in figure 12.

As shown in the figure 12, for the first run the input file is divided into seven chunks. All four I frames are covered completely in these chunks and carving process would recover all four I frames. In
the second run the chunk size is increased and now the input file is divided into four chunks. Three of
the I frames are completely present in the chunks where as the second I frame is split half in the first
and second chunks which are then placed at different locations on the data set. Carving in this
scenario would recover three complete I frames and one partial I frame or may not recover the second
I frame at all if at least one slice is not present after the frame. The graph in figure 11 shows this
scenario clearly where at some points number of fully recovered I frames decreases even if the chunk
size is increased.

The second data set contains an elementary stream MPEG-1 file with resolution 320*240, frame rate
25 frames per second, and bit rate 0.58 Mbit/s. The original file size is 3593 KB and it has total 41 I
frames. There are total 899 (round figure) blocks in the file. For each run on the data set we chose a
random entry point block 565 that results in 335 remaining blocks. These 335 blocks contain 16 I
frames in total. Table 4 shows the percentage of recovered frames for each of the runs.

<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>335</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>75</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>112</td>
<td>6.25</td>
<td>0</td>
<td>12.5</td>
<td>81.25</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>67</td>
<td>43.75</td>
<td>0</td>
<td>18.75</td>
<td>37.5</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>48</td>
<td>62.5</td>
<td>0</td>
<td>12.5</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>38</td>
<td>62.5</td>
<td>0</td>
<td>6.25</td>
<td>31.25</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>31</td>
<td>56.25</td>
<td>0</td>
<td>25</td>
<td>18.75</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>26</td>
<td>81.25</td>
<td>0</td>
<td>12.5</td>
<td>6.25</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>23</td>
<td>68.75</td>
<td>0</td>
<td>6.25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>20</td>
<td>93.75</td>
<td>0</td>
<td>0</td>
<td>6.25</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>18</td>
<td>93.75</td>
<td>0</td>
<td>6.25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>16</td>
<td>93.75</td>
<td>0</td>
<td>0</td>
<td>6.25</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>15</td>
<td>81.25</td>
<td>0</td>
<td>6.25</td>
<td>12.5</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>14</td>
<td>87.5</td>
<td>0</td>
<td>12.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>13</td>
<td>93.75</td>
<td>0</td>
<td>0</td>
<td>6.25</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>12</td>
<td>81.25</td>
<td>0</td>
<td>6.25</td>
<td>12.5</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>11</td>
<td>87.5</td>
<td>0</td>
<td>12.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>11</td>
<td>68.75</td>
<td>0</td>
<td>12.5</td>
<td>18.75</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>10</td>
<td>87.5</td>
<td>0</td>
<td>6.25</td>
<td>6.25</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Carving results of second data set

The third data set contains an elementary stream MPEG-1 file with resolution 352*288, frame rate
29.9 frames per second, and bit rate 1.125 Mbit/s. The original file size is 17335 KB and it has in total
249 I frames. There are 4334 (round figure) blocks in the file. For each run on the data set we chose a random entry point block 738 that results in 3597 remaining blocks. These 3597 blocks contain 212 I frames in total. Table 5 shows the percentage of recovered frames for each of the runs.

<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3597</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>1199</td>
<td>15.09</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>84.91</td>
</tr>
<tr>
<td>20</td>
<td>720</td>
<td>44.34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>55.66</td>
</tr>
<tr>
<td>28</td>
<td>514</td>
<td>60.85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>39.15</td>
</tr>
<tr>
<td>36</td>
<td>400</td>
<td>70.75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>29.25</td>
</tr>
<tr>
<td>44</td>
<td>327</td>
<td>74.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25.94</td>
</tr>
<tr>
<td>52</td>
<td>277</td>
<td>74.06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25.94</td>
</tr>
<tr>
<td>60</td>
<td>240</td>
<td>83.49</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16.51</td>
</tr>
<tr>
<td>68</td>
<td>212</td>
<td>86.32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13.68</td>
</tr>
<tr>
<td>76</td>
<td>190</td>
<td>88.21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11.79</td>
</tr>
<tr>
<td>84</td>
<td>172</td>
<td>87.26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12.74</td>
</tr>
<tr>
<td>92</td>
<td>157</td>
<td>86.79</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13.21</td>
</tr>
<tr>
<td>100</td>
<td>144</td>
<td>86.32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13.68</td>
</tr>
<tr>
<td>108</td>
<td>134</td>
<td>88.21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11.79</td>
</tr>
<tr>
<td>116</td>
<td>125</td>
<td>92.45</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7.55</td>
</tr>
<tr>
<td>124</td>
<td>117</td>
<td>91.98</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8.02</td>
</tr>
<tr>
<td>132</td>
<td>109</td>
<td>89.15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10.85</td>
</tr>
<tr>
<td>140</td>
<td>103</td>
<td>92.45</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7.55</td>
</tr>
</tbody>
</table>

Table 5: Carving results of third data set

The fourth data set contains an elementary stream MPEG-1 file with resolution 352*288, frame rate 25 frames per second, and bit rate 2.92 Mbit/s. The original file size is 46594 KB and it has 230 I frames. With an input block size of 4 KB there are total 11649 (round figure) blocks in the file. For each run on the data set we chose a random entry point block 11079 that results in 571 remaining blocks. These 571 blocks contain 15 I frames in total. Table 6 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>571</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>191</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>115</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>28</td>
<td>82</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>36</td>
<td>64</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>44</td>
<td>52</td>
<td>46.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53.33</td>
</tr>
<tr>
<td>52</td>
<td>44</td>
<td>46.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53.33</td>
</tr>
<tr>
<td>60</td>
<td>39</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>68</td>
<td>34</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>76</td>
<td>31</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>84</td>
<td>28</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>92</td>
<td>25</td>
<td>66.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.33</td>
</tr>
<tr>
<td>100</td>
<td>23</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>108</td>
<td>22</td>
<td>73.33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>26.67</td>
</tr>
<tr>
<td>116</td>
<td>20</td>
<td>66.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.33</td>
</tr>
<tr>
<td>124</td>
<td>19</td>
<td>86.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13.33</td>
</tr>
<tr>
<td>132</td>
<td>18</td>
<td>86.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13.33</td>
</tr>
<tr>
<td>140</td>
<td>17</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 6: Carving results for fourth data set

The fifth data set contains an elementary stream MPEG-1 file with resolution 640*480, frame rate 25 frames per second, and bit rate 3.51 Mbit/s. The original file size is 8828 KB and it has 50 I frames. With an input block size of 4 KB there are total 2207 blocks in the file. For each run on the data set we chose a random entry point block 1451 that results in 757 remaining blocks. These 757 blocks contain 24 I frames in total. Table 7 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>757</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>253</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>152</td>
<td>16.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>83.33</td>
</tr>
<tr>
<td>28</td>
<td>109</td>
<td>33.33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66.67</td>
</tr>
<tr>
<td>36</td>
<td>85</td>
<td>45.83</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>54.17</td>
</tr>
<tr>
<td>44</td>
<td>69</td>
<td>95.83</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.17</td>
</tr>
<tr>
<td>52</td>
<td>59</td>
<td>66.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.33</td>
</tr>
<tr>
<td>60</td>
<td>51</td>
<td>75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>68</td>
<td>45</td>
<td>62.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37.5</td>
</tr>
<tr>
<td>76</td>
<td>40</td>
<td>66.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.33</td>
</tr>
<tr>
<td>84</td>
<td>37</td>
<td>79.17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20.83</td>
</tr>
<tr>
<td>92</td>
<td>33</td>
<td>75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>100</td>
<td>31</td>
<td>83.33</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16.67</td>
</tr>
<tr>
<td>108</td>
<td>29</td>
<td>75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>116</td>
<td>27</td>
<td>91.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8.33</td>
</tr>
<tr>
<td>124</td>
<td>25</td>
<td>87.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12.5</td>
</tr>
<tr>
<td>132</td>
<td>23</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.17</td>
</tr>
<tr>
<td>140</td>
<td>22</td>
<td>91.67</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8.33</td>
</tr>
</tbody>
</table>

**Table 7: Carving results for fifth data set**

Figure 13 shows the graph representation of carving results for the first five data sets. In the first three data sets larger number of I frames are completely recovered in smaller chunk size as compared to the fourth and fifth data sets. This is because the input MPEG-1 files in the first three data sets have low bit rate values as compared to those used in fourth and fifth data sets. The size of a compressed video increases with the increase of bit rate. This means frame sizes will also increase and therefore more chunk size would be required to carve the frames successfully.

The highest mean value for the first five data sets is equal to 87.802 % when the chunk size is 124 KB. The quantity of 124 KB is roughly equal to the average size of a GOP plus the average size of two I frames. This is the minimum size of file fragments on a disk having elementary stream MPEG-1 video data with a chance to recover 87.802 % I frames completely. The rest of the I frames will be recovered either partially or ignored completely.
The sixth data set contains a system stream MPEG-1 file with resolution 320*240, frame rate 29.9 frames per second, and bit rate 1.13 Mbit/s. The original file size is 22167 KB and it has 302 I frames. With an input block size of 4 KB there are total 5542 blocks in the file. For each run on the data set we chose a random entry point block 587 that results in 4956 remaining blocks. These 4956 blocks contain 270 I frames in total. Table 8 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100%)</th>
<th>Partially Recovered (99%-90%)</th>
<th>Semi Recovered (89%-50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4956</td>
<td>0.37</td>
<td>0</td>
<td>0.37</td>
<td>99.26</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1652</td>
<td>2.22</td>
<td>0</td>
<td>38.15</td>
<td>59.63</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>992</td>
<td>33.33</td>
<td>0.37</td>
<td>27.78</td>
<td>38.52</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>708</td>
<td>52.59</td>
<td>0.37</td>
<td>20.74</td>
<td>26.3</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>551</td>
<td>62.59</td>
<td>0</td>
<td>17.41</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>451</td>
<td>70.74</td>
<td>0.37</td>
<td>11.85</td>
<td>17.04</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>382</td>
<td>73.33</td>
<td>0.37</td>
<td>12.22</td>
<td>14.07</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>331</td>
<td>75.93</td>
<td>0</td>
<td>8.52</td>
<td>16.3</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>292</td>
<td>80.74</td>
<td>0.37</td>
<td>10.37</td>
<td>8.52</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>261</td>
<td>81.48</td>
<td>0</td>
<td>8.89</td>
<td>9.63</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>236</td>
<td>83.33</td>
<td>0</td>
<td>7.78</td>
<td>8.89</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>216</td>
<td>84.81</td>
<td>0</td>
<td>5.19</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>199</td>
<td>85.56</td>
<td>0.37</td>
<td>6.3</td>
<td>7.78</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>184</td>
<td>87.04</td>
<td>0</td>
<td>5.93</td>
<td>7.04</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>171</td>
<td>86.67</td>
<td>0</td>
<td>7.04</td>
<td>6.3</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>160</td>
<td>89.63</td>
<td>0</td>
<td>4.81</td>
<td>5.56</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>151</td>
<td>90.37</td>
<td>0.37</td>
<td>3.33</td>
<td>5.93</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>142</td>
<td>88.89</td>
<td>0</td>
<td>5.56</td>
<td>5.56</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8: Carving results for sixth data set

The seventh data set contains a system stream MPEG-1 file with resolution 352*240, frame rate 25 frames per second, and bit rate 1.70 Mbit/s. The original file size is 32625 KB and it has 246 I frames. With an input block size of 4 KB there are total 8157 blocks in the file. For each run on the data set we chose a random entry point block 7501 that results in 657 remaining blocks. These 657 blocks contain 20 I frames in total. Table 9 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>657</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>75</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>219</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>75</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>132</td>
<td>35</td>
<td>0</td>
<td>15</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>28</td>
<td>94</td>
<td>25</td>
<td>5</td>
<td>20</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>73</td>
<td>45</td>
<td>0</td>
<td>30</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>60</td>
<td>55</td>
<td>5</td>
<td>35</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>51</td>
<td>80</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>44</td>
<td>80</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>39</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>35</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>32</td>
<td>75</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>29</td>
<td>75</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>27</td>
<td>85</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>25</td>
<td>85</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>23</td>
<td>75</td>
<td>10</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>22</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>20</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>19</td>
<td>85</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 9: Carving results for Seventh data set**

The eighth data set contains a system stream MPEG-1 file with resolution 352*240, frame rate 25 frames per second, and average bit rate 4.09 Mbit/s. The original file size is 17283 KB and it has 64 I frames. With an input block size of 4 KB there are total 4321 blocks in the file. For each run on the data set we chose a random entry point block 3872 that results in 450 remaining blocks. These 450 blocks contain 7 I frames in total. Table 10 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>450</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>71.43</td>
<td>28.57</td>
</tr>
<tr>
<td>12</td>
<td>150</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>85.71</td>
<td>14.29</td>
</tr>
<tr>
<td>20</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>42.86</td>
<td>57.14</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>65</td>
<td>0</td>
<td>0</td>
<td>57.14</td>
<td>42.86</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>50</td>
<td>28.57</td>
<td>0</td>
<td>42.86</td>
<td>28.57</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>41</td>
<td>57.14</td>
<td>14.29</td>
<td>28.57</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>35</td>
<td>42.86</td>
<td>0</td>
<td>42.86</td>
<td>14.29</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>30</td>
<td>57.14</td>
<td>0</td>
<td>14.29</td>
<td>28.57</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>27</td>
<td>14.29</td>
<td>14.29</td>
<td>42.86</td>
<td>28.57</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>24</td>
<td>85.71</td>
<td>14.29</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>22</td>
<td>71.43</td>
<td>0</td>
<td>14.29</td>
<td>14.29</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>20</td>
<td>57.14</td>
<td>0</td>
<td>28.57</td>
<td>14.29</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>18</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>17</td>
<td>57.14</td>
<td>0</td>
<td>14.29</td>
<td>28.57</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>16</td>
<td>71.43</td>
<td>0</td>
<td>14.29</td>
<td>14.29</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>15</td>
<td>71.43</td>
<td>0</td>
<td>14.29</td>
<td>14.29</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>14</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>13</td>
<td>71.43</td>
<td>0</td>
<td>28.57</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 10: Carving results for eighth data set

The ninth data set contains a system stream MPEG-1 file with resolution 352*288, frame rate 25 frames per second, and bit rate 2.37 Mbit/s. The original file size is 15360 KB and it has 80 I frames. With an input block size of 4 KB there are total 3840 blocks in the file. For each run on the data set we chose a random entry point block 3088 that results in 753 remaining blocks. These 753 blocks contain 18 I frames in total. Table 11 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>753</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>61.11</td>
<td>38.89</td>
</tr>
<tr>
<td>12</td>
<td>251</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>83.33</td>
<td>16.67</td>
</tr>
<tr>
<td>20</td>
<td>151</td>
<td>5.56</td>
<td>5.56</td>
<td>0</td>
<td>83.33</td>
<td>5.56</td>
</tr>
<tr>
<td>28</td>
<td>108</td>
<td>5.56</td>
<td>0</td>
<td>38.89</td>
<td>55.56</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>84</td>
<td>27.78</td>
<td>5.56</td>
<td>22.22</td>
<td>44.44</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>69</td>
<td>22.22</td>
<td>5.56</td>
<td>27.78</td>
<td>44.44</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>58</td>
<td>38.89</td>
<td>0</td>
<td>38.89</td>
<td>16.67</td>
<td>5.56</td>
</tr>
<tr>
<td>60</td>
<td>51</td>
<td>50</td>
<td>5.56</td>
<td>16.67</td>
<td>27.78</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>45</td>
<td>55.56</td>
<td>0</td>
<td>22.22</td>
<td>22.22</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>40</td>
<td>61.11</td>
<td>0</td>
<td>16.67</td>
<td>22.22</td>
<td>0</td>
</tr>
<tr>
<td>84</td>
<td>36</td>
<td>94.44</td>
<td>0</td>
<td>5.56</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>33</td>
<td>77.78</td>
<td>0</td>
<td>11.11</td>
<td>11.11</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>31</td>
<td>72.22</td>
<td>0</td>
<td>0</td>
<td>27.78</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>28</td>
<td>77.78</td>
<td>0</td>
<td>16.67</td>
<td>5.56</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>26</td>
<td>66.67</td>
<td>0</td>
<td>11.11</td>
<td>16.67</td>
<td>5.56</td>
</tr>
<tr>
<td>124</td>
<td>25</td>
<td>88.89</td>
<td>0</td>
<td>11.11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>23</td>
<td>77.78</td>
<td>5.56</td>
<td>5.56</td>
<td>11.11</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>22</td>
<td>77.78</td>
<td>0</td>
<td>5.56</td>
<td>16.67</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 11: Carving results for ninth data set

The tenth data set contains a system stream MPEG-1 file with resolution 640*480, frame rate 29.9 frames per second, and bit rate 3.05 Mbit/s. The original file size is 5881 KB and it has 29 I frames. With an input block size of 4 KB there are total 1471 blocks in the file. For each run on the data set we chose a random entry point block 800 that results in 672 remaining blocks. These 672 blocks contain 13 I frames in total. Table 12 shows the percentage of recovered frames for each of the runs.
<table>
<thead>
<tr>
<th>Chunk size in KB</th>
<th>Number of chunks</th>
<th>Fully Recovered (100 %)</th>
<th>Partially Recovered (99% - 90%)</th>
<th>Semi Recovered (89% - 50%)</th>
<th>Less than 50% recovered</th>
<th>Frames not recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>672</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46.15</td>
<td>53.85</td>
</tr>
<tr>
<td>12</td>
<td>224</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>92.31</td>
<td>7.69</td>
</tr>
<tr>
<td>20</td>
<td>135</td>
<td>0</td>
<td>0</td>
<td>7.69</td>
<td>92.31</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>30.77</td>
<td>61.54</td>
<td>7.69</td>
</tr>
<tr>
<td>36</td>
<td>75</td>
<td>23.08</td>
<td>0</td>
<td>15.38</td>
<td>53.85</td>
<td>7.69</td>
</tr>
<tr>
<td>44</td>
<td>62</td>
<td>46.15</td>
<td>0</td>
<td>15.38</td>
<td>38.46</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
<td>15.38</td>
<td>0</td>
<td>0</td>
<td>76.92</td>
<td>7.69</td>
</tr>
<tr>
<td>60</td>
<td>45</td>
<td>46.15</td>
<td>0</td>
<td>23.08</td>
<td>30.77</td>
<td>0</td>
</tr>
<tr>
<td>68</td>
<td>40</td>
<td>38.46</td>
<td>0</td>
<td>30.77</td>
<td>30.77</td>
<td>0</td>
</tr>
<tr>
<td>76</td>
<td>36</td>
<td>69.23</td>
<td>0</td>
<td>0</td>
<td>23.08</td>
<td>7.69</td>
</tr>
<tr>
<td>84</td>
<td>32</td>
<td>69.23</td>
<td>0</td>
<td>15.38</td>
<td>15.38</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>30</td>
<td>69.23</td>
<td>0</td>
<td>7.69</td>
<td>23.08</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>27</td>
<td>76.92</td>
<td>0</td>
<td>7.69</td>
<td>15.38</td>
<td>0</td>
</tr>
<tr>
<td>108</td>
<td>25</td>
<td>69.23</td>
<td>0</td>
<td>7.69</td>
<td>23.08</td>
<td>0</td>
</tr>
<tr>
<td>116</td>
<td>24</td>
<td>76.92</td>
<td>0</td>
<td>15.38</td>
<td>7.69</td>
<td>0</td>
</tr>
<tr>
<td>124</td>
<td>22</td>
<td>76.92</td>
<td>0</td>
<td>15.38</td>
<td>7.69</td>
<td>0</td>
</tr>
<tr>
<td>132</td>
<td>21</td>
<td>84.62</td>
<td>0</td>
<td>0</td>
<td>15.38</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>20</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 12: Carving results for tenth data set

Figure 14 shows the graph representation of carving results for the last five data sets. The highest mean value in this case is equal to 90.55 % when the chunk size is 132 KB. As we can see the required chunk size value is increased in the case of system stream MPEG-1 files. As shown in figure 5, I frame has a different structure when part of a system stream. The frame data is divided into several PES packets each having its own header. There will be an arbitrary number of PES headers and Pack headers present within the frame boundary with a possibility of audio frame data as well. This would result in larger chunk size required to fully carve I frames as compared to the case of the elementary stream MPEG-1 data.
Figure 14: Chunk size and number of fully recovered I frames in last five data sets
7 Conclusion and Future Work

7.1 Concluding Remarks

File carving is a difficult task and does not guarantee 100% recovery of deleted files even if deleted data is not overwritten. We introduce the concept of stream carving and discuss how streaming multimedia formats are suitable for carving due to the nature of their file structure. After recovering a video file from seized storage device it has to be analyzed manually in order to see if it contains any illegal contents. Since video files may consist of hundreds of frames depending on their length, manual screening is time consuming and intensive and significantly prolongs the duration of the forensic analysis.

To overcome this limitation, we propose a carving approach for streaming video formats that can be used to automate analysis of recovered video files. Using this approach we developed a carving tool that recovers I (key) frames of deleted MPEG-1 files. The recovered frames are then stored as image files. Forensic analysis of these images now can be performed automatically using several approaches. The tool displays all recovered frames, which also allows the examiner to manually examine each frame.

MPEG-1 is a complex format and difficult to handle due to its variable nature. An MPEG-1 file may contain only video data or both audio and video data. In both cases structure of the stream is different and both types of files can be present in the disk at the same time. The first objective of the carver was to handle both formats at the same time that is achieved successfully. Another objective was to decode carved frames for which we need to locate correct frame resolution. In section 4.2.1 we explain that for each frame the carving algorithm searches backward in a specific threshold range to locate sequence header and then reads the resolution information. We also explain that the carver locates all possible unique resolution values in the raw data set and decodes a frame with all these values where sequence header is not found. This technique works well most of the times, but there may be problems with certain files as discussed in section 4.2.1. Therefore, there is a potential for improvement of the current approach to handle this issue in a better manner with respect to correctness and accuracy.

For any state-of-the-art carving tool it is important to be able to handle fragmented files. During the experiments we simulated a highly fragmented environment. The results of these experiments show that our approach is successful in such cases. The main advantage of our approach is that we target individual frames and start carving at the beginning of the frame header. This means even if the original MPEG-1 file is divided into several fragments stored sequentially or non-sequentially, we can still carve individual frames as long as the frame is present in its entirety.

We also show how partial frames can be carved using a simple approach although it was not our objective to handle fragmentation inside the boundary of a frame. As shown in the experimental results even when large fragmentation is present, we were able to fully recover most of the I frames. In real-life scenarios a file is commonly stored in two or three fragments since most of the current operating systems are designed to avoid fragmentation as much as possible. Therefore, we can safely assume that in real scenarios the percentage of fully recovered I frames will be reasonably higher.
7.2 Future Work

Our approach can be extended to other file formats having similar structure. MPEG-2 format is very close to MPEG-1 in terms of structure. There are few additional things added in MPEG-2 format but the basic structure remains the same. One major difference is that MPEG-2 supports a broad range of commonly used frame resolutions unlike MPEG-1 that uses three (SIF) resolutions. For MPEG-2, this results in a large number of possible unique resolution values in the data set. Therefore it is not suitable to decode a frame with all these resolutions in case we are not able to locate sequence header with that frame and read the resolution information. A different technique is required to be developed in this case in order to efficiently locate the frame resolution values.

Certain improvements can also be made in the carving algorithm. We provided a basic solution to handle fragmentation inside a frame’s boundary. Using this we are only able to recover partial frames but could not locate all fragments of a frame and rearrange the fragments in order to reconstruct a complete frame. To handle fragmentation at this level we need to incorporate block content based carving approach instead of relying only on the file structure information. The information presented in this thesis can be used to develop a block content based carving approach for MPEG-1 videos.
References


37. ISO/IEC 13818-2, Information Technology-Generic Coding of Moving Pictures and Associated Audio Recommendation H.262


44. Java MPEG-1 Video Decoder and Player, http://sourceforge.net/projects/javaMPEG1video/


Appendix A

Bash script to store MPEG-1 file on raw data set

#!/bin/bash

infile="name and path of input MPEG-1 file"
outfile="name and path of output raw data set"
COUNT=1 # Increment by 2 with each run
SKIP=X   # X = entry point -1
SEEK=100
loop=Y   # number of chunks
while [ $loop -gt 0 ]
do
    dd if=$infile skip=$SKIP of=$outfile bs=4K count=$COUNT seek=$SEEK
    conv=notrunc,noerror
    SKIP=($($SKIP+$COUNT))
    SEEK=($($SEEK+300))
    let loop=loop-1
done
## Appendix B

### Definition of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra quantization matrix</td>
<td>A set of values used for quantization of I frames.</td>
</tr>
<tr>
<td>Non intra quantization matrix</td>
<td>A set of values used for quantization of B and P frames.</td>
</tr>
<tr>
<td>Quantization</td>
<td>Is one of the compression techniques used for MPEG-1 videos.</td>
</tr>
<tr>
<td>Quantizer scale factor</td>
<td>A value between 1 to 31 that defines the amount of compression to achieve.</td>
</tr>
<tr>
<td>Time code</td>
<td>Contains time stamp value for the first frame in a GOP.</td>
</tr>
</tbody>
</table>