Decision-theoretic file carving

Pavel Gladyshev[a], Joshua Isaac James[b]

[a]Digital Forensics Investigation Research Laboratory, University College Dublin, Belfield, Dublin 4, Ireland
[b]Digital Forensics Investigation Research Laboratory, Hallym University, 1 Hallimdaehak-gil, Chuncheon-si, Gangwon-do, South Korea

Abstract

This article explores a novel approach to file carving by viewing it as a decision problem. It allows us to design algorithms that produce best effort results under the given resource constraints. It is important for digital forensic triage as well as for e-discovery. As an illustration, we develop a novel file carving algorithm together with a novel linear-time detector of JPEG data and examine its performance.

Keywords: decision-theoretic file carving, file carving, digital forensic investigation, Digital forensic triage, preliminary analysis

1. Introduction

Digital forensic investigations must process large amounts of evidential data with limited resources (Casey et al., 2009; Pollitt, 2013). Modern solutions to the data processing problem combine advances in automatic detection of relevant data with some form of selective human exploration to identify sample data and to validate output results (Marturana and Tacconi, 2013; Schell et al., 2007; James et al., 2014). Despite significant time and cost savings that automation can bring to digital investigations, this approach still relies on exhaustive processing of the suspect data.

Various methods have been proposed that attempt to prioritise exhibits by first prioritising suspect data sources that the exhibit contains (Shaw and Browne, 2013; Rogers, 2003; Overill et al., 2013). Several works (Koopmans and James, 2013; Casey et al., 2009) have discussed the idea of digital forensic triage; an investigation processes with the goal of not always requiring an exhaustive search of all devices, -and helps with decisions of relevancy and prioritisation.

There are many situations where a digital investigator is limited either in time or available computational power. For example, a parole officer may use a portable forensic tool to periodically check that an offender's computer does not contain child abuse material. The time and processing power available to the officer on site is limited and – given the growth in data storage capacity – may not be sufficient to exhaustively explore the contents of an offender's computer. Thus a solution is required that would combine automatic processing with probabilistic sampling and prioritisation aimed at discovering relevant information more quickly.

This paper explores the idea of probabilistic sampling and prioritisation in the context of file carving. It shows that when the investigator is looking for files of a particular kind, such as large JPEG files, this information can be used to speed up file carving by prioritizing processing of data blocks that are most likely to contain relevant data. This is demonstrated by constructing specialised file carver for digital photographs that outperforms[1] traditional file carvers.

1.1. Contribution

This work contributes to the field of digital forensic investigation by proposing a *best-effort* file carving method that aims to recover as much relevant information as possible under the given time- and processing power constraints. It is suitable for digital forensic triage purposes as defined by (Koopmans and James, 2013). Specifically, this work provides:

- a formal statement of file carving as decision problem;
- an illustration of how simplifies mathematical models and simulation can help improving file carving efficiency;
- a JPEG image carving algorithm and software application that could recover potentially relevant images (photographs) faster than traditional carvers.

2. Sequential file carving

File carving is often used in digital investigations. Traditional file carving programs, like `foremost` by (Richard III and Roussev, 2005) or `photorec` by (Grenier, 2007) can be described as *sequential* file carvers. The basic sequential carving algorithm works as follows: the carving starts at the first data block of the drive and progresses consecutively until the end of the drive is reached. Each block is checked against a database of header and footer signatures. If the block matches some *header* signature, it is assumed to be the first block of a file to be recovered. The rest of the file is assumed to occupy the subsequent consecutive blocks up to and including the matching "footer"

Email address: `pavel.gladyshev@ucd.ie` (Pavel Gladyshev)

[1]The relevant measure of performance is discussed in Section 3.2.1

block. If the "footer" block cannot be found, the length of the recovered file is capped at a predefined limit.

Sequential file carving works because the mainstream file systems are designed to store file data in consecutive disk blocks whenever possible, in order to maximize performance of mechanical hard disk drives. While file fragmentation does occur, (Garfinkel, 2007) shows that the majority of files in personal computers are not fragmented.

Prior research in file carving mainly focused on improving accuracy; more accurate techniques for detecting file fragments of particular kinds were proposed[2], and algorithms for detection and reassembly of fragmented files were developed that achieve quadratic time complexity[3].

The performance of the basic sequential carving algorithm was improved upon by Richard, et al (Richard et al., 2007), and similarly Meijer (Meijer, Rob, 2012), who proposed in-place[4] file carving in an attempt to lower the time and space requirements compared to traditional sequential carving. These approaches identify the challenges with traditional file carving, and propose methods to reduce carving resource requirements by minimizing the amount of disk I/O associated with copying recovered file data. Essentially, once file data is discovered, pointers are created in virtual file systems that point to the data directly on the suspect disk rather than copying data from the suspect disk. However, the approach used for *finding* the file data is still sequential, as before.

3. File carving as a Battleships game

File carving is a process of exploration, and it is instructive to consider file carving as an analogy of the Battleships[5] game, in which one player – the data storage – arranges her ships (the relevant files) on a "canal" and the other player – the file carver – tries to destroy the opponent's ships with the minimal number of bombs (see Fig. 1).

The simplest strategy that is guaranteed to destroy the opponent's fleet is to bomb all squares consecutively from left to right, which is precisely what sequential carvers do. Although effective, this strategy is not necessarily the most efficient. A human players of Battleships would randomly bomb the opponent's map until one of the ships is hit. After that, she would bomb the adjacent squares until the damaged ship is completely destroyed. It is intuitively plausible that similar approach could be used in file carving and that it may outperform sequential carving algorithms. In the rest of this paper we present theoretical and experimental results that validate this intuition using ecision theory, numeric simulation, and actual file carving experiments.

3.1. Decision theory basics

Decision theory is a branch of mathematics that studies decision making. In its basic form, it represents decision as a choice between several alternative actions that together form the set A (possible actions):

$$A = \{a_1, a_2, \ldots, a_n\} \tag{1}$$

Any chosen action $a_i \in A$ can result in one of several possible, mutually exclusive outcomes that together form the set O (possible outcomes):

$$O = \{o_1, o_2, \ldots o_m\} \tag{2}$$

The likelihood of a particular outcome $o_j \in O$ happening as a result of the action $a_i \in A$ is represented by the conditional probability $P(o_i \mid a_i)$.

The desirability of a particular outcome is modelled by the *utility* function $u()$ that assigns a real-valued score to each outcome. The higher the utility score, the more desirable is the outcome. The cost associated with doing a particular action can be modeled as a reduction of utility scores of the corresponding outcomes. To model it formally we define $u()$ as function of both the outcome *and* the action:

$$u : O \times A \to \mathbb{R} \tag{3}$$

The principle of *Maximum Expected Utility* (MEU) states that a rational decision maker should choose the action with the highest mathematical expectation of the utility score across all possible outcomes:

$$a = \underset{a_i \in A}{\operatorname{argmax}} \sum_{o_j \in O} u(o_j, a_i) P(o_j \mid a_i) \tag{4}$$

3.2. File carving as decision problem

Like a player of Battleships that chooses which square to bomb, file carver chooses which disk block to process next. Let array B represent the drive subjected to file carving:

$$B = (b_0, b_1, \ldots b_m) \tag{5}$$

Each element $b_0 \ldots b_m$ represents a separate data block on the drive.

Let file carving *action* be the act of processing a particular disk block $b_k \in B$. There are two possible outcomes: b_k contains the relevant data (r), or b_k does not contain relevant data (n):

$$O = \{r, n\} \tag{6}$$

[2] For more detail see (Veenman, 2007), (Li et al., 2011), and (Garfinkel and McCarrin, 2015)

[3] $O(n^2)$ time complexity where n is the number of fragments to be reassembled. For more detail see (Memon and Pal, 2006)

[4] Also known as zero-storage

[5] The Battleships game is played as follows: each player has a pad of paper with two 100-square grids (maps) – one for arranging the player's ships and the other for noting the discovered locations of the opponent's ships. Before the start of the game, each player secretly arranges their ships on their grid. A ship occupies a number of consecutive squares and is oriented either vertically or horizontally. The ships cannot overlap. The game proceeds in turns with each player sending a "bomb" to a particular square on the enemy grid. The other player then responds whether the "bombing" succeeded in damaging one of their ships. A ship is destroyed when all of the squares it occupies are hit by the opponent. The game ends when one of the players completely destroys the other player's fleet.

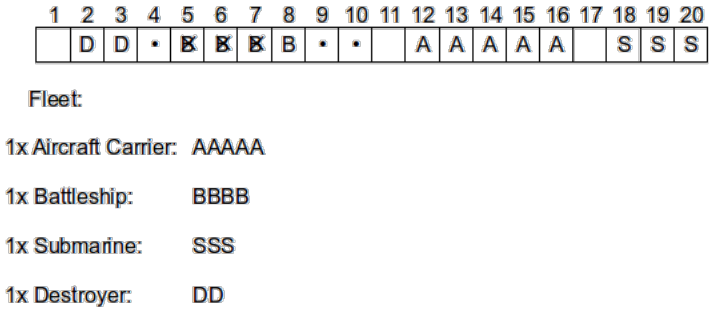

Figure 1: A 1-dimensional Battleships game.

Given that hard disk drives have physical read-write heads, that may have to be repositioned to access a different disk block, it is convenient to *refer* to a particular block by its *offset* from the last processed block. We will use such offsets instead of block indices to identify actions that can be taken by the carver:

$$A = \{\ldots, -3, -2, -1, 1, 2, 3 \ldots\} \tag{7}$$

For example, if the last processed block was b_{10}, the action $a = -3$ means examination of the block b_7.

3.2.1. Utility of file carving actions

In triage situations it is important to get relevant files quickly. Sequential carving takes long time to find relevant files if they are spread throughout the disk space. Figure 2 shows a plot of the total amount of data recovered by a sequential carver as function of time. Steep climbs occur when the carver finds relevant data; long horisontal stretches occur when the carver scans through drive areas devoid of relevant data. Figure 2 clearly shows that utility of carving actions can be defined as the *slope* of that curve:

$$u(o, a) = \begin{cases} 0 & o = n \\ \frac{1}{T_{proc}(a)} & o = r \end{cases} \tag{8}$$

where

$$T_{proc}(a) = t_{RAM} + t_{seek}(a) \tag{9}$$

Definition (8) consists of two parts. The first part of it says that processing blocks containing irrelevant data has zero utility in the sense that it does not increase the total amount of recovered data. The second part states that the usefulness of processing a block of relevant data is determined by how fast we process it – the slower we process it, the more gentle is the slope of the curve in Figure 2, and lower the utility of it.

$T_{proc}(a)$ is the time required to process a disk block. It consists of a relatively constant component t_{RAM} – the time required to read a block of data into RAM and process it there, and $t_{seek}(a)$ – the drive seek time[6] that generally depends on how far the target data block is from the last processed block. If the block at the offset a contains relevant data, its utility is the reciprocal of $T_{proc}(a)$.

The substitution of (8) for $u(o, a)$ in (4) produces the deicsion principle for file carving:

$$a = \underset{a \in A}{\operatorname{argmax}} \left(\frac{1}{T_{proc}(a)} p(a, l) \right) \tag{10}$$

where function $p(a, l) = P(o = r \mid a, b_l)$ represents the probability of finding a relevant data block at a particular offset a from the last processed block b_l. Parameter l is the index of the last processed block: $0 \leq l \leq m$. Function $p(a, l)$ represents the current "knowledge" of the carver of where the relevant blocks are likely to be and serves as a scaling factor reducing the benefit of processing a particular block if that block is unlikely to contain relevant data.

To implement a carver based on the Equation (10), the following things are required:

1. a reliable and efficient method to detect *relevant* data blocks. Unlike the header and footer signatures used in sequential carving, this method should work for arbitrary blocks from the middle of the file;
2. an understanding of the shape and properties of $p(a)$ in the context of a particular carving task, and how $p(a)$ changes when some block is examined and found to be either relevant or irrelevant;
3. an efficient way to determine $\operatorname{argmax}_{a \in A} \left(\frac{1}{T_{proc}(a)} p(a, l) \right)$ at run time.

It is also important to understand and practically demonstrate the circumstances in which decision-theoretic file carver would outperform sequential file carver. To prove the viability of decision-theoretic carving we developed a specialised file carver for recovering large JPEG files.

[6] the time taken for a disk drive to locate the area on the disk where the data to be read is stored.

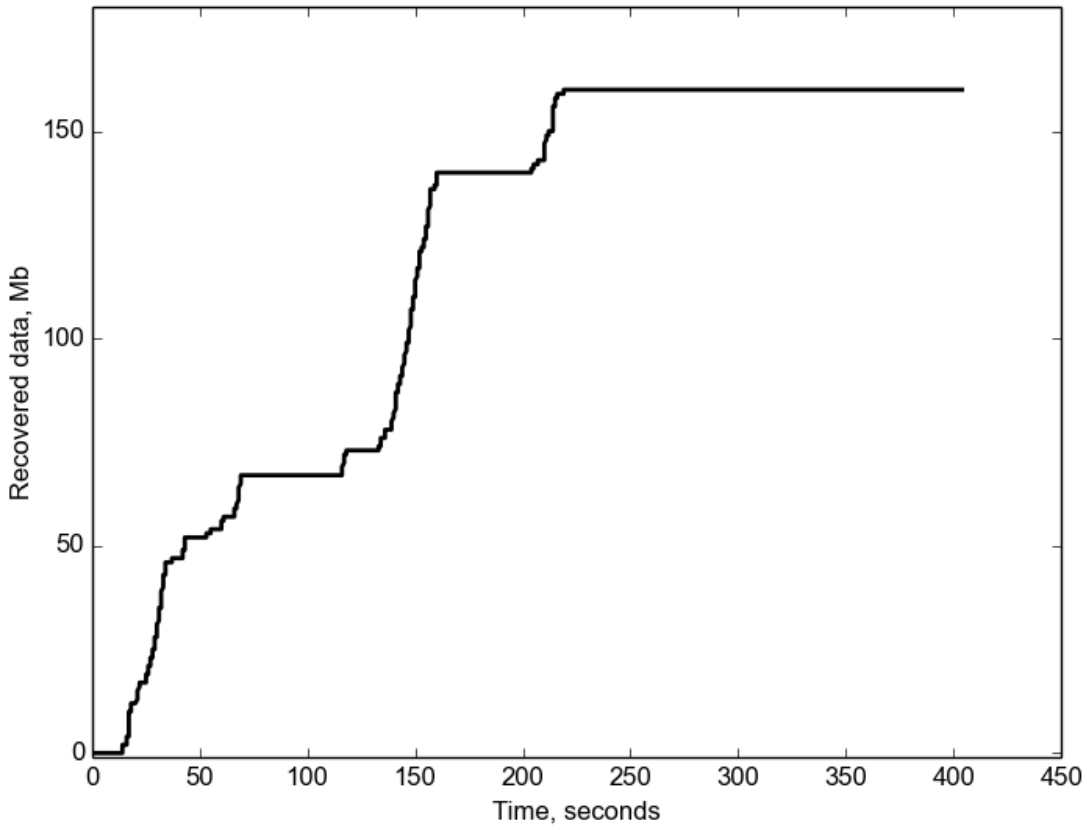

Figure 2: The total amount of data recovered by a sequential file carver as a function of time. In this example the relevant data (JPEG images) occupied approximately 5% of a 32Gb disk image.

4. The need for selective recovery of large JPEG files

Suppose that parole officer needs to check the computer of a sex offender for presence of raw digital photographs that may indicate production of child abuse material. To do it, she or he needs a program that can quickly find as many raw digital photographs as possible. The computer in question may not have much processing power, and the time for analysis may be limited. Decision-theoretic approach fits this problem very well, because it focuses on making best decision at every step leading to the best-effort results under given circumstances.

A digital camera sensor typically produces raw photographs at a fixed resolution. The raw sensor data is then resized and compressed according to the image quality settings specified by the user. The resulting JPEG file is, typically, several megabytes in size.

Figure 3 shows a histogram of sizes of digital photographs from a personal collection of 5000 digital photographs accumulated over 10 years. There are several peaks corresponding to different cameras and image quality settings used over the years. Each peak resembles the "bell curve" of the normal distribution[7]. Note that the vast majority of JPEG files in Figure 3 are bigger than 700 Kb in size. The two right-most peaks in the histogram correspond to images produced by a newer camera, whose JPEG files a bigger than 2000 Kb. It is highly likely that mainstream camera resolution will continue to grow leading to larger size of JPEG files.

5. Detector of JPEG data

A fast and accurate detector of relevant data is essential for efficient operation of decision-theoretic file carver. Our initial implementation of JPEG data detector used Support Vector Machine (SVM) classifier based on the ideas of Li *et al.* described in (Li et al., 2011). The detector used 256 data features corresponding to byte-value frequencies. The detector was trained using 17,918 training samples that contained 8,985 samples from known Non-JPEG files and 8,933 samples from known JPEG files. It included samples from all sections of JPEG files. After training the detector was tested on a much larger data sample derived from a different source. It demonstrated 99% true positive rate and 33% false positive rate.

The high false positive rate is due to relatively high entropy of compressed JPEG data. According to our tests the trained linear SVM classifier did not distinguish JPEG data from other high-entropy data, such as ZIP files. As we shall see in Section 10, the high false positive rate of the detector can make decision-theoretic carving highly inefficient. In a nutshell, SVM-based detector forced our carver to spend a lot of time looking for JPEGs in non-JPEG high-entropy files (ZIP, GIF, RAR files, etc.). As a result, our carver behaved more like a typical sequential carver. The development of efficient and accurate detector of relevant data is, therefore, essential problem in decision-theoretic carving.

Clearly, an accurate detector of relevant data can be constructed only if the target data has features that reliably distinguishes it from any other data. Even if such features are present, it may not be possible to compute them efficiently[8]. Since the aim of this paper was to *demonstrate* viability of the decision-theoretic carving, we developed a simple *ad-hoc* detector for JPEG data that provided acceptable precision and performance characteristics for our experiments (Algorithm 1).

Our detector relies on the presence of escape sequences in JPEG data. Escape sequences are used by JPEG encoder for in-band signalling. When a byte value of 255 is encountered in the JPEG stream, JPEG decoder treats is as the beginning of a command. The subsequent bytes specify what the JPEG decoder must do. For disambiguation purposes any actual 255 data value is encoded as 255 followed by 0. Our detector defines JPEG data as any high entropy data that contains at least one JPEG escape sequence. Instead of calculating the actual entropy measure, it constructs a histogram of byte values from the given data block and checks that no single value occurs exceedingly often.

Observe that Algorithm 1 is quite efficient. It has linear time complexity with respect to the size of d. This is similar to the complexity of signature-based detection methods used in mainstream file carvers.

6. Probability of finding JPEG data in a particular block on the drive

The second problem that needs to be solved to create a decision-theoretic JPEG carver is to determine the probability of finding a piece of JPEG digital photograph in a particular data block on the suspect hard disk drive. For the purposes of this article we decided to build a simplified mathematical model of the JPEG carving problem and explore its properties using small-scale simulation.

We make the following core assumptions about JPEG files:

1. JPEG files are not fragmented;
2. JPEG files do not overlap and do not share blocks;
3. the only other information we have about the target files is the minimal and maximal file size limits specified in the file carver configuration settings.

According to the principle of uninformed priors (Jaynes, 2003), we ought make the following additional assumptions:

5. JPEG file sizes are uniformly distributed between minimal and maximal limits;
6. starting locations of JPEG files are uniformly distributed throughout the disk space subject to non-fragmentation and non-overlapping requirements;

[7] Recall that JPEG encoding divides the image into square fragments 8×8 pixels. Every such fragment is compressed independently meaning that by the central limit theorem of probability theory, the probability distribution of JPEG sizes for a particular camera setting converges to normal (Gaussian) distribution.

[8] Consider, for example, the problem of selectively carving child pornography images from unallocated disk space

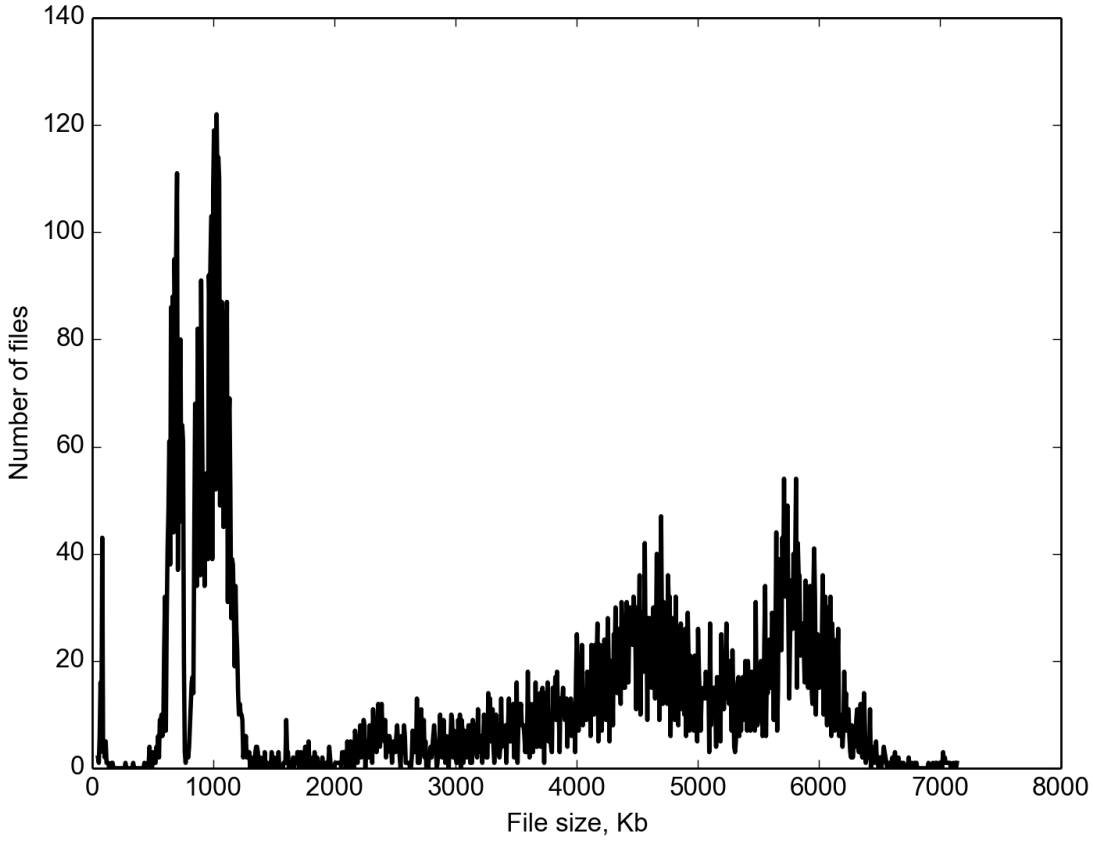

Figure 3: Histogram of actual file sizes from a personal collection of raw digital photographs in JPEG format. Horizontal axis is the size of files in Kibibytes, vertical axis is the number of files. Each of the peaks visible in the histogram corresponds to a particular camera and image size and quality settings.

Algorithm 1 Simple detector of JPEG data (*JpegDetect*)

INPUT:
array $d[N]$ of N eight-bit bytes.
RETURNS:
true if d contains JPEG data, false otherwise.
VARIABLES:
integer array $c[256]$ for counting distinct byte values in d,
integer variable esc for counting JPEG escape sequences.

```
esc ← 0                                          ▷ Initialize counters
for i = 0 to 255 do
    c[i] ← 0
end for

for i = 0 to N − 2 do                            ▷ Iterate over d
    val ← d[i]
    nextval ← d[i + 1]

    c[val] ← c[val] + 1

    if c[val] > 50 then
      return false    ▷ all values should be equally infrequent
    end if

    if val = 255 and nextval = 255 then
      return false          ▷ A 255 must not be followed by 255
    end if

    if val = 255 and nextval ∈ {0, 208, 209, . . . , 215} then
        esc ← esc + 1
    end if
end for

if esc > 0 then
      return true           ▷ Must contain at least one escape seq.
else
      return false
end if
```

7. starting locations of JPEG files are uniformly distributed throughout the disk space subject to non-fragmentation and non-overlapping requirements;
8. the total number of blocks occupied by relevant data is not known and the probability that the relevant data occupies a certain amount of disk space is uniformly distributed between 0 and 100 percent.

6.1. Estimating probability function with small-scale simulation

To get a sense of the shape and properties of The probability of encountering JPEG data in a particular block $P(o = r|b_i)$, we performed a series of small-scale simulations using a "toy" model of a hard disk drive with only 20 blocks: $B = (b_0, \ldots, b_{19})$. The model represents each block with a single value: the value of 1.0 corresponds to blocks containing JPEG data while 0.0 represents blocks with irrelevant information. The small number of blocks in the model allowed brute-force computation of $P(o = r|b_i)$ by constructing all possible "contents" of the drive satisfying the assumptions about the location and sizes of JPEG files. The value of $P(o = r|b_i)$ for a particular block b_i is calculated as the frequency of b_i containing JPEG data across all constructed disk contents[9].

We began our analysis by considering a very simple case: the drive contains single fixed-size JPEG file. The results of the small scale simulation are shown in Figure 4. The plot of $P(o = r|b_i)$ has trapezoidal shape. To see why it is so, consider Figure 5. Under the assumption of non-fragmentation, an 11-block continuous JPEG file can reside in one of 10 possible positions on a 20-block drive. Observe that two blocks in the middle of the drive are occupied in every possible position of the file, while the very first and the very last blocks on the drive are occupied only in one of ten possible positions of the file. That results in a trapezoidal form of the probability distribution $P(o = r|b_i)$ seen in Figure 4.

Another case we considered is when the drive contains several non-fragmented files of the same size. Figure 6 shows the results of the corresponding small scale simulation. In this example we reduced the fixed length of JPEG file down to 6 blocks, and our 20-block drive can now contain up to three such files. For the sake of clarity we added the third dimension to the plot: the total amount of JPEG data blocks stored on the drive. It allows us clearly see the probability $P(o = r|b_i)$ when the drive has either 1, 2 or 3 JPEG files stored on it. To better understand the shapes of these probability plots, the following flashlight analogy could be helpful: imagine an electric flashlight that takes three D-type batteries. A single battery can slide freely inside the flashlight. If we find a flashlight with a single battery inside laying on the floor, the exact position of the battery inside the case is somewhat uncertain to us, which corresponds to the first trapezoidal probability "ridge" in Figure 6. If, on the other hand, we know that all three batteries are in the flashlight, then we are quite certain where they are inside the flashlight. It corresponds to the last row of three probability "ridges" in 6, which are considerably higher than the first.

Note also, that if we see someone else's flashlight on the floor and have no prior information about the number of batteries in it, we ought to assume all possibile battery configurations equally likely, and base our decision making on the probability plot that is the average of probability plots for all possible number of batteries inside.

The final case we considered was the drive that contains a number of JPEG files above certain size. The corresponding simulation results are shown in Figure 7 and Figure 8. The latter is the average probability plot assuming that all possible file configurations on the drive are equally likely.

A notable feature of the Figure 8 is the presence of two peaks that make it look somewhat like a cat's head. Each of the peaks is located at the offset corresponding to the minimal file length away from each end of the drive. These peaks offer a slightly

[9]The corresponding python script `simulator.py` can be found in the source code of our carver (Gladyshev and James, 2016).

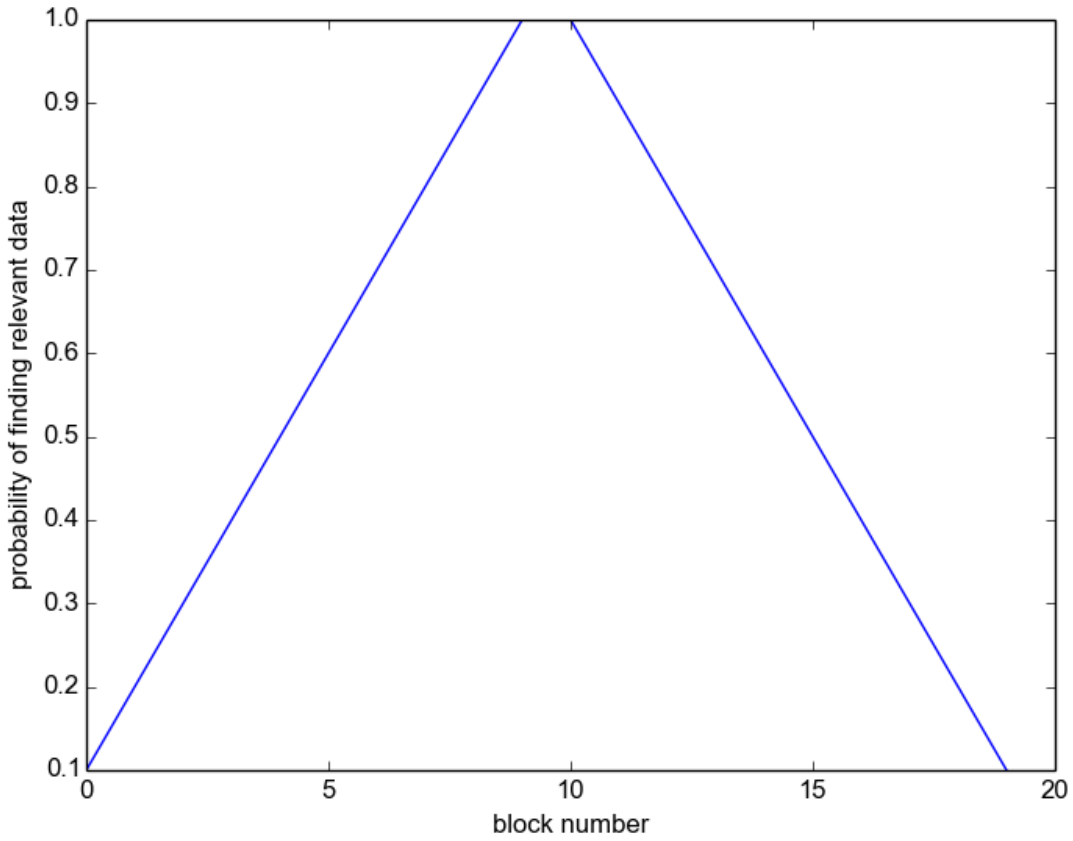

Figure 4: Probability of finding relevant data in a particular disk block, assuming that JPEG files are *exactly* 11 blocks in size.

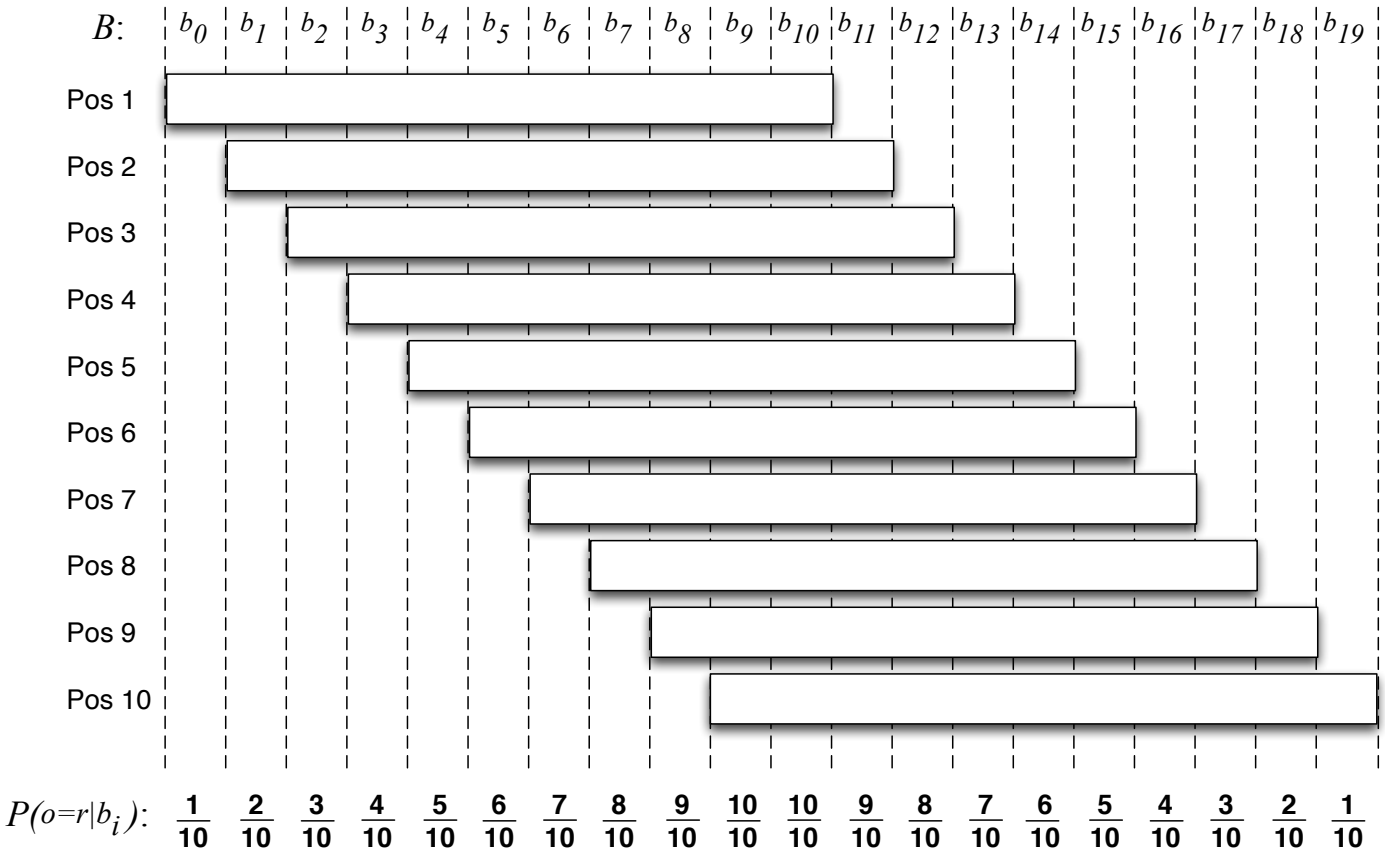

Figure 5: Explanation of the trapezoidal form of the $p(a)$ probability distribution in Figure 4.

higher probability of finding JPEG data there, than on the rest of the drive, and – if $\frac{1}{T_{proc}(a)}$ is constant – these peaks are the optimal choices for the decision-theoretic block testing according to the Equation (10).

6.2. Simulating results of successful and unsuccessful block testing

Our model can also be used to explore the result of testing a particular block on the drive and finding either relevant or irrelevant data there. This is achieved by restricting probability calculation to only those drive configurations that either do or do not contain relevant data in the tested block.

Figure 9 shows the result of testing the drive block b_5 corresponding to the first probability peak in Figure 8 and finding no JPEG data there. Observe that the negative test at b_5 implies that blocks $b_0 \ldots b_4$ also do not contain JPEG data. It is the consequence of assuming a lower bound of 6 blocks on JPEG file size together with the absence of fragmentation.

Note also that the shape of the posterior probability distribution to the right of b_5 is quite similar to the form of the initial probability distribution from Figure 8. The difference is that the first peak of probability is now at the block b_{11} instead of b_5. This is again a direct consequence of the non-fragmentation assumption and the minimal file size assumption.

The implication of the Figure 9 is that we may not always need to check every block on the drive to find all JPEGs above certain size. In the small-scale model from Figure 9 it suffices to check only three blocks: b_5,b_{11}, and b_{14} to establish that there is no JPEGs on the 20-block drive! The larger the minimal size of JPEG files, the less blocks we need to check.

Figure 10 shows the result of testing drive block b_5 and finding that it contains relevant data. As expected, positive test increases the probability of finding JPEG data at b_5 to 1.0. Note that the next most likely blocks to contain JPEG data are blocks b_4 and b_6 adjacent to b_5. This result is in full accordance with the human intuition of how to play the Battleships game: once an enemy ship is discovered by random bombing, human player finishes it by bombing squares adjacent to the first successful hit.

7. Exploring properties of $T_{proc}(a)$ for hard disk drives and solid state drives.

The final piece of knowledge that needs to be acquired in order to build a decision-theoretic carver is the form and properties of $T_{proc}(a)$. In order to better understand its properties for actual HDD and SSD devices, we performed a series of experiments on a 120Gb Fujitsu HDD and a 120Gb SSDNow SSD.

Two types of experiments were performed: one group of experiments measured $T_{proc}(a)$ for random block access, and the other group of experiments measured sequential access time. Given below is the summary of our findings.

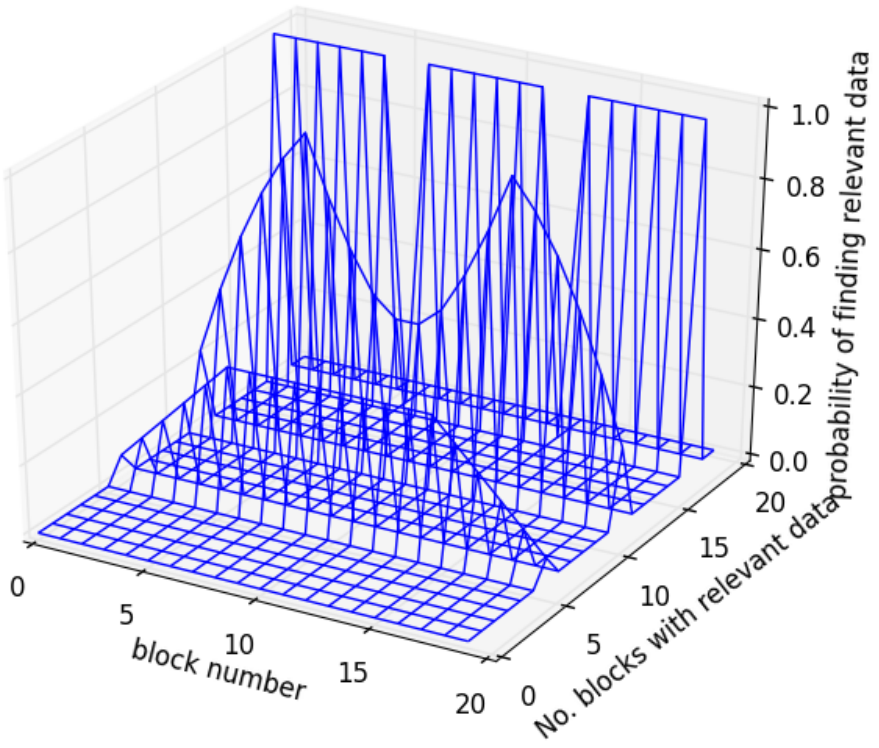

Figure 6: Probability of finding relevant data in a particular disk block, given that certain number of blocks are occupied by relevant data and assuming that JPEG files are *exactly* 6 blocks in size.

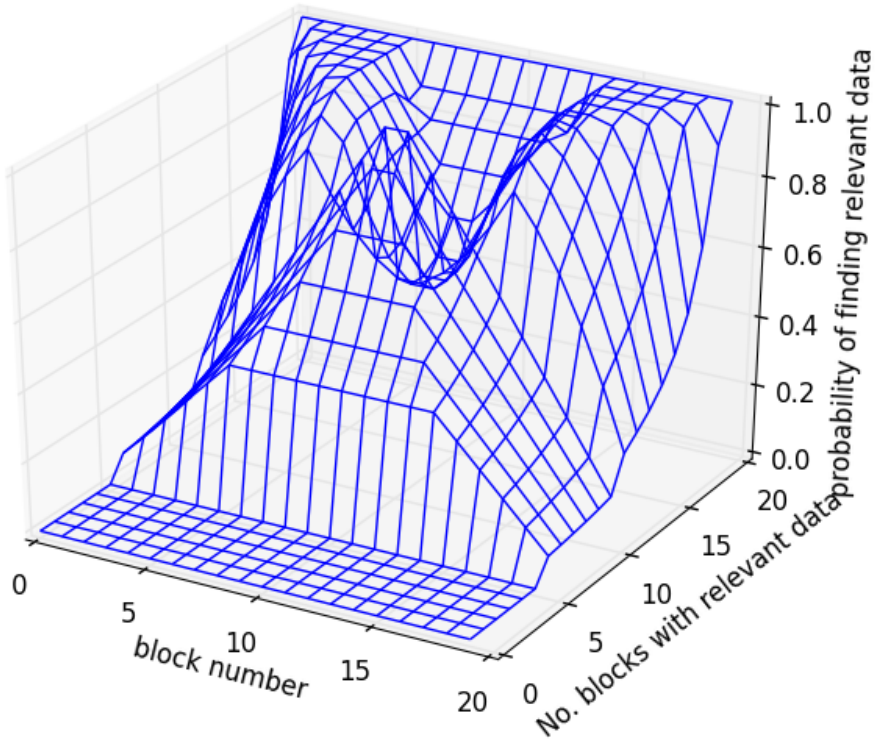

Figure 7: Probability of finding relevant data in a particular disk block, given that certain number of blocks are occupied by relevant data and assuming that JPEG files are *at least* 6 blocks in size.

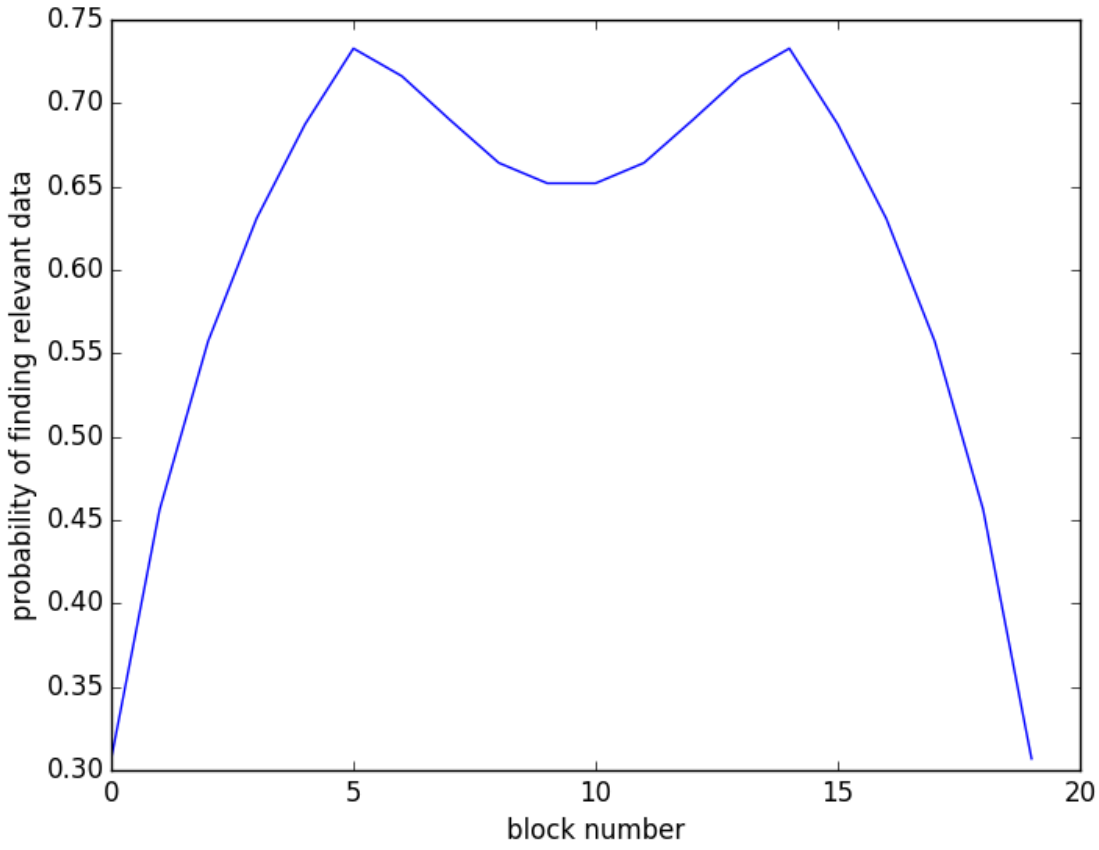

Figure 8: Probability of finding relevant data in a particular disk block, assuming that JPEG files are *at least* 6 blocks in size.

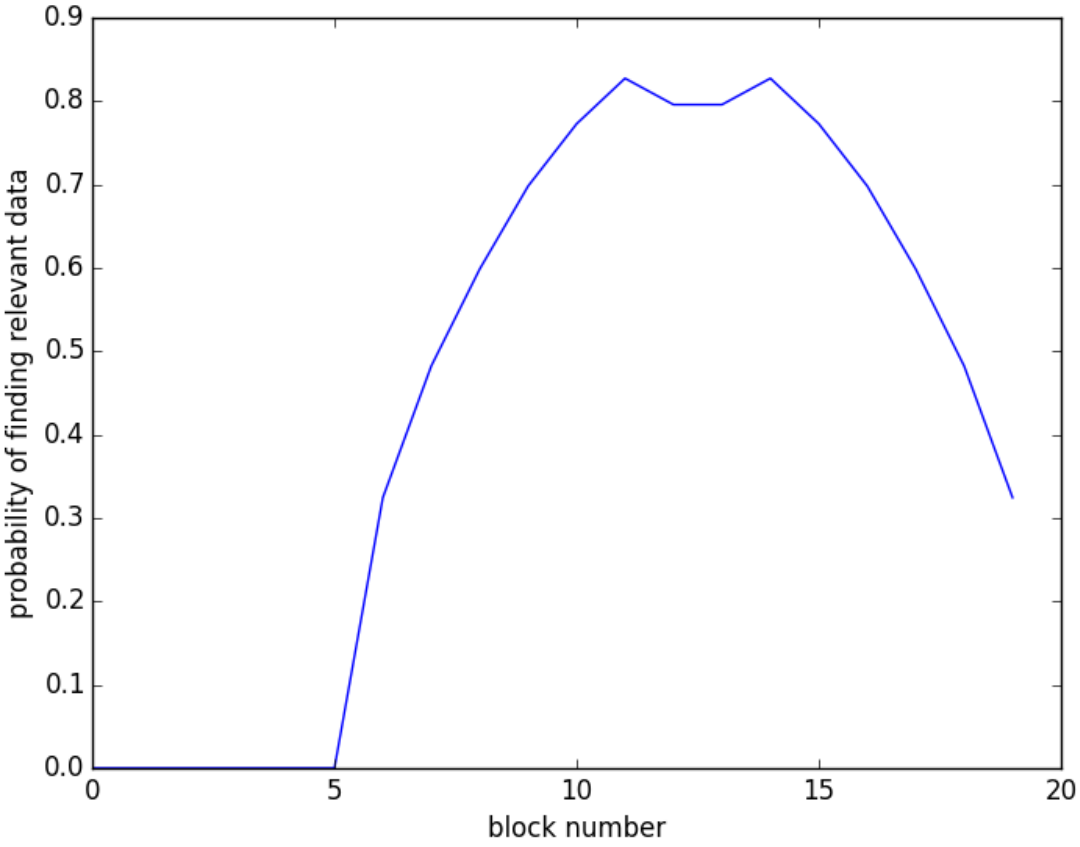

Figure 9: Posterior probability of finding JPEG data in a particular disk block, after testing b_5 and finding no relevant JPEG data there. All JPEG files are assumed to be at least 6 blocks in length.

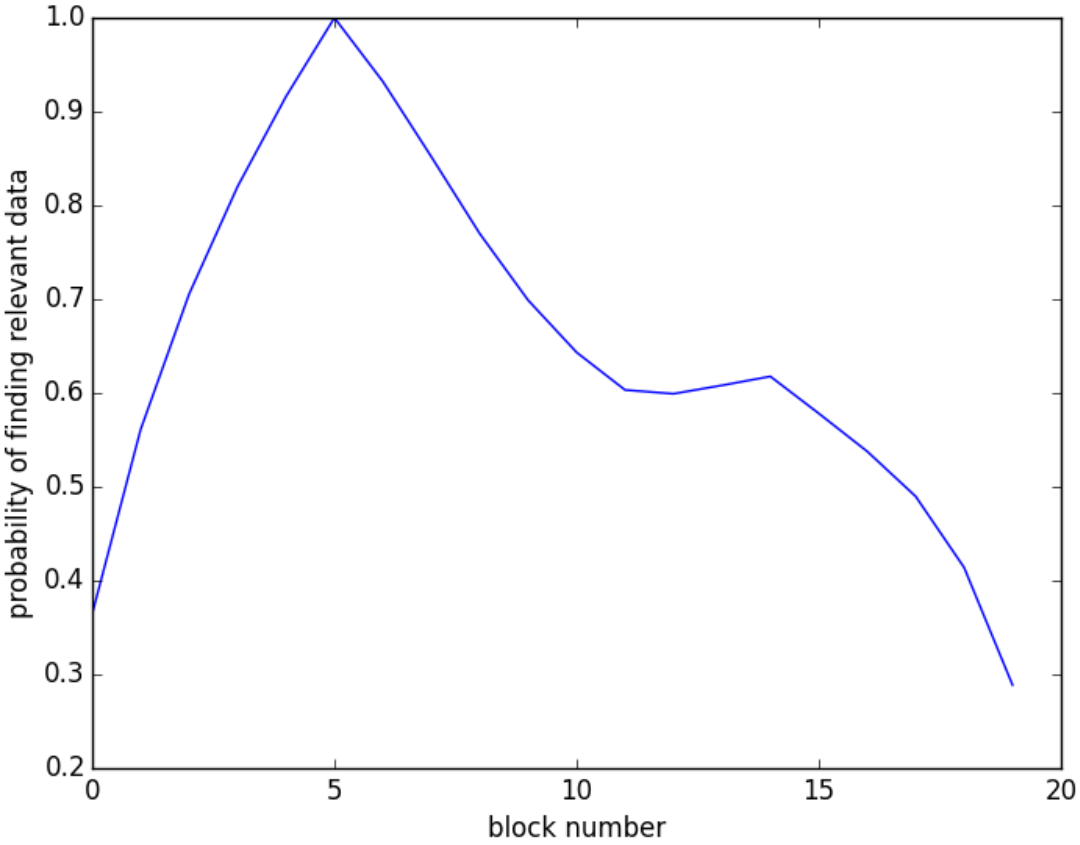

Figure 10: Posterior probability of finding JPEG data in a particular disk block, after testing b_5 and finding that it contains JPEG data. All JPEG files are assumed to be at least 6 blocks in length.

To measure the time required for random block access, 1,000 random block tests were performed for each drive. Each random block test consists of a drive seek operation followed by reading the target disk block and performing JPEG data detection on the read data using Algorithm 1. The time required to complete each experiment was measured. We denote this time as $T_{rbt}(a)$. Figures 11 and 12 show the empirical $T_{rbt}(a)$ values for the 120Gb HDD and 120Gb SSD respectively.

The experimental data is quite noisy, which is probably due to RAM caching of the previously read blocks and fluctuations of normal system activity. Nevertheless, it suggests linear relationship between the absolute seek distance $|a|$ and processing time:

$$T_{rbt}(a) = \beta|a| + \gamma \tag{11}$$

where β and γ can be estimated using linear regression of the experimental data. The resulting linear approximations are shown in Figures 11 and 12 as dashed lines. We observe that in both cases $T_{rbt}(a)$ has relatively large constant component(γ) and very shallow slope (β). In SSD experiments β is almost zero and $T_{rbt}(a)$ is essentially a constant over the entire range of seek offsets. In HDD experiments $T_{rbt}(a)$ clearly increases proportionally to the seek offset, but despite a pronounced visual appearance in Figure 11, the sloping angle of $T_{rbt}(a)$ is actually so shallow and that it can be ignored at short travel distances like the size of a large JPEG file (see footnote[10] for explanation).

Although random block access is important for decision-theoretic carving, we should not overlook properties of sequential block access on SSD and HDD. It is well known that both SSDs and HDDs are optimised for sequential block access, and it is much quicker to read 100 blocks sequentially rather than randomly. Suppose that our file carver needs to examine content of a block at a (positive) offset a from the last examined block. This can be done either by a direct jump followed by the block processing or by reading and processing consecutive blocks up to and including the target block. To distinguish the two approaches we denote the time required to read and examine a consecutive blocks sequentially as $T_{seq}(a)$.

To determine how these approaches compare, we conducted additional experiments measuring the time taken to read and apply signature matching to large stretches of consecutive blocks on the 120Gb Fujitsu HDD and the 120Gb SSDNow SSD. As expected, $T_{seq}(a)$ grows at a rate proportional to a for both SSD and HDD. To compare the behaviour of $T_{rbt}(a)$ and $T_{seq}(a)$, we plotted them together for the SSD (Figure 13) and HDD (Figure 14) respectively.

It is easy to see that $T_{seq}(a)$ is smaller for small values of a, while $T_{rbt}(a)$ is smaller for large values of a. The point of equivalence is e where

$$T_{rbt}(e) = T_{seq}(e) \tag{12}$$

It is hardware-dependent and in our experiments it varied between 10^2 and 10^3 4Kb blocks for SSDs and between 10^4 and 10^5 4Kb blocks for HDDs.

8. Designing new carving strategy for large, non-fragmented JPEG files

Based on the results of Sections 5, 6 and 7 we are now in a position to propose a new strategy for carving large non-fragmented JPEG files.

We first go back to the decision equation (10) and substitute the right-hand side of 11 for $T_{proc}(a)$ in it:

$$a = \underset{a_i \in A}{\operatorname{argmax}} \left(\frac{1}{\beta a_i + \gamma} p(a_i, l) \right) \tag{13}$$

As demonstrated in the Figure 12, SSDs have constant $T_{proc}(a_i)$. It effectively turns $\frac{1}{\beta a_i + \gamma}$ into a constant scaling factor $\frac{1}{\gamma}$, and the shape of the utility expression is determined entirely by the shape of $p(a_i, l)$.

For HDDs the effect of T_{proc} on the decision making is more pronounced. Since T_{proc} increases proportionally to the jump distance a_i, the multiplier $\frac{1}{\beta a_i + \gamma}$ decreases inversely proportional to a_i. The net effect of this is that out of two blocks with equal $p(a_i, l)$, Equation (10) would favour the block closest to the previously examined block. Due to extremely shallow slope, however, the effect of utility reduction is significant only at values of a_i that are many times greater than the size of typical JPEG file. At values of a_i comparable with the size of a typical JPEG file, we can assume $\frac{1}{\beta a_i + \gamma}$ to be constant, and the shape of the utility expression in Equation (13) determined entirely by the shape of $p(a_i, l)$ as in the SSD case.

The results presented in Section 6 indicate that the probability function $p(a_i, l)$ has two peaks: one at the distance $L_{min} - 1$ from the current position on the drive, where L_{min} is the minimal size of JPEG file in blocks, and the other peak is at the distance $L_{max} - (L_{min} - 1)$, where L_{max} is the size of the unexplored portion of data blocks adjacent to the last examined block. For SSDs both locations are *equally* optimal, because T_{proc} is constant, but for HDDs the location $L_{max} - (L_{min} - 1)$ is clearly sub-optimal, if L_{max} is large. In either case, *choosing to examine disk block at the offset $L_{min} - 1$ blocks from the last examined block seems to be the optimal decision when either initiating the carving process or after an unsuccessful block test.*

It is easy to see that for relatively small JPEG files, whose sizes are below the intersection point e of $T_{rbt}(a)$ and $T_{seq}(a)$, sequential carving *is* the optimal strategy[11], because $T_{seq}(a)$ is the fastest way to "sample" and extract all blocks at a_i distances equal to the size of such files. For solid state drives these are

[10] According to the estimates shown in Figure 11, HDD seek time increases from around 15 ms for short jumps to aproximately 32 ms for jumps across the entire 120 Gb of the HDD. Most JPEGs are under 50Mb in size. At such small distances the increase of $T_{rbt}(a)$ would be only 0.007 ms or 0.0005% of the constant 15 ms. Thus, for short jumps comparable to the size of JPEG file, we can assume $T_{rbt}(a)$ to be constant.

[11] Note that while sequential carving is optimal for recovering *all* small files, it may not always be necessary to recover all of them. In triage situations we may be content with just a subset of JPEGs, in which case random access testing may still produce a speed-up.

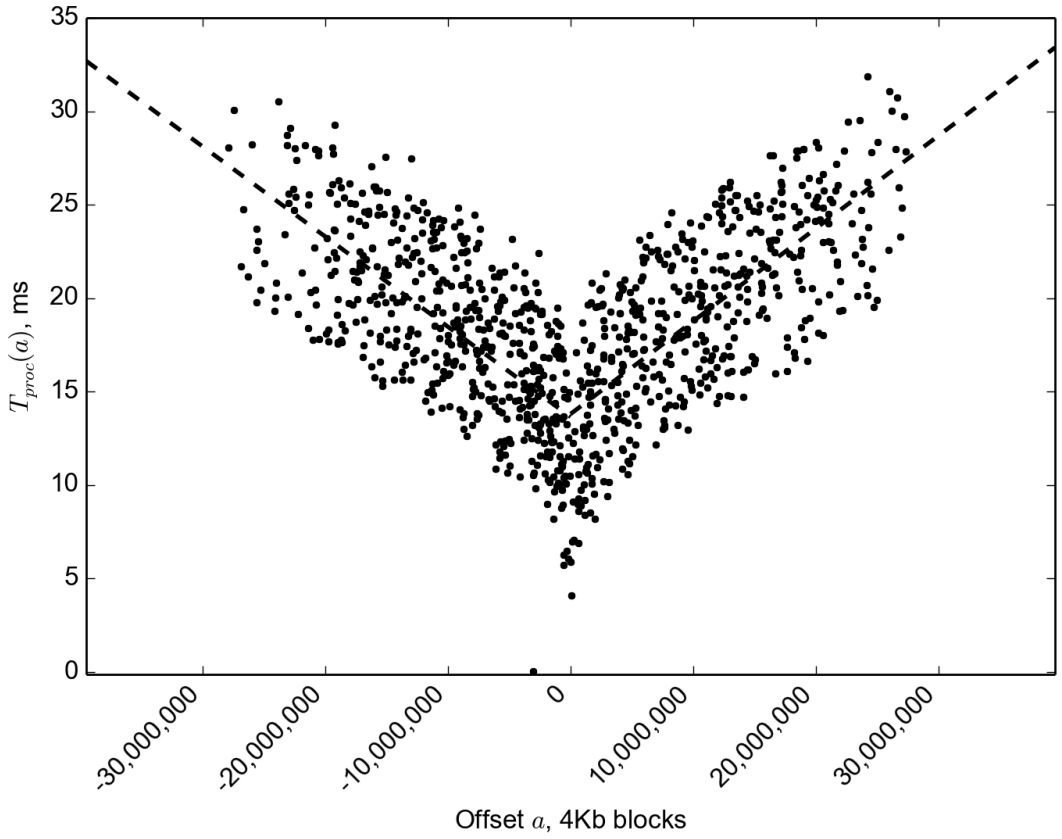

Figure 11: Time of random block testing on Fujitsu 120 Gb HDD.

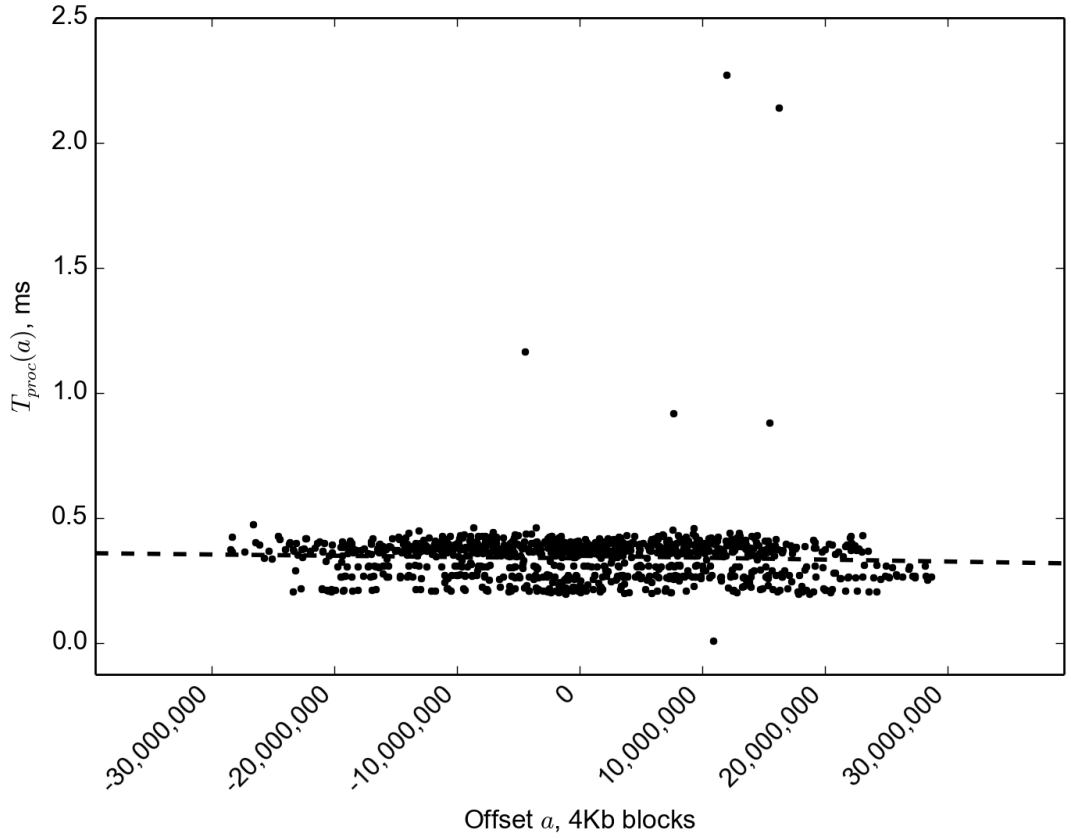

Figure 12: Time of random block testing on SSDNow 120 Gb SSD.

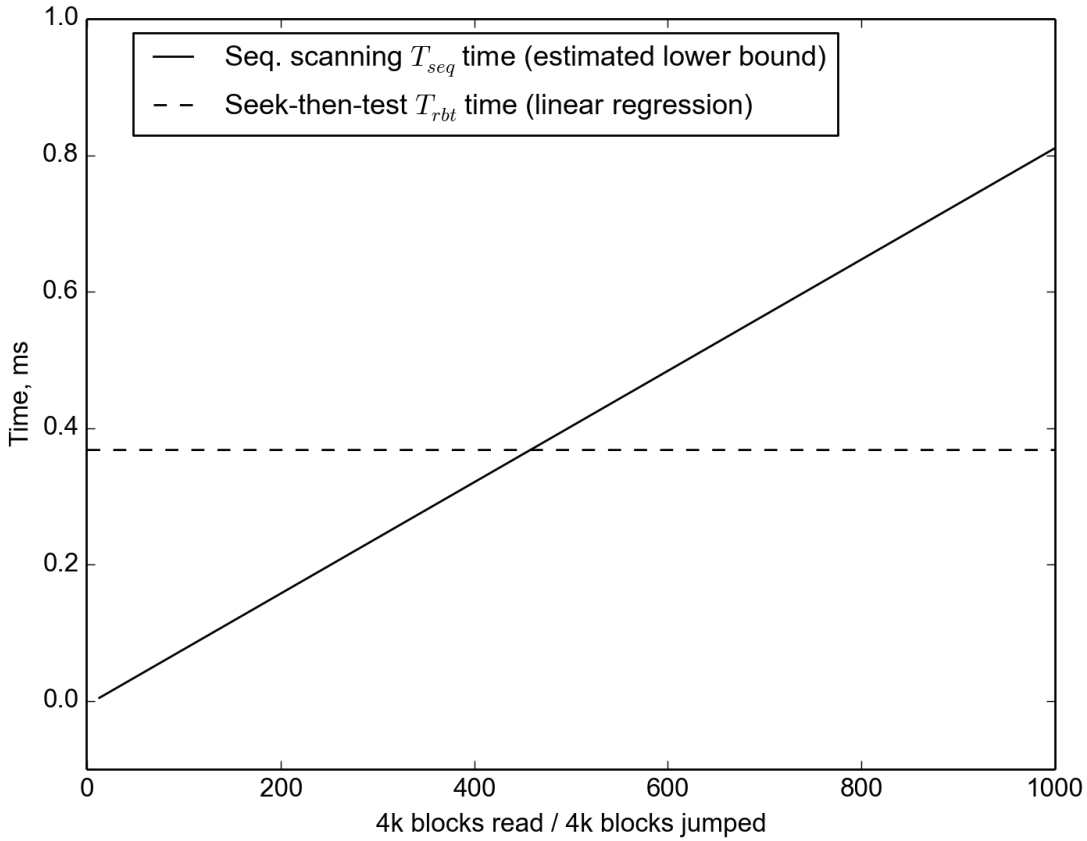

Figure 13: $T_{rbt}(a)$ and $T_{seq}(a)$ calculated for SSDNow 120 Gb SSD.

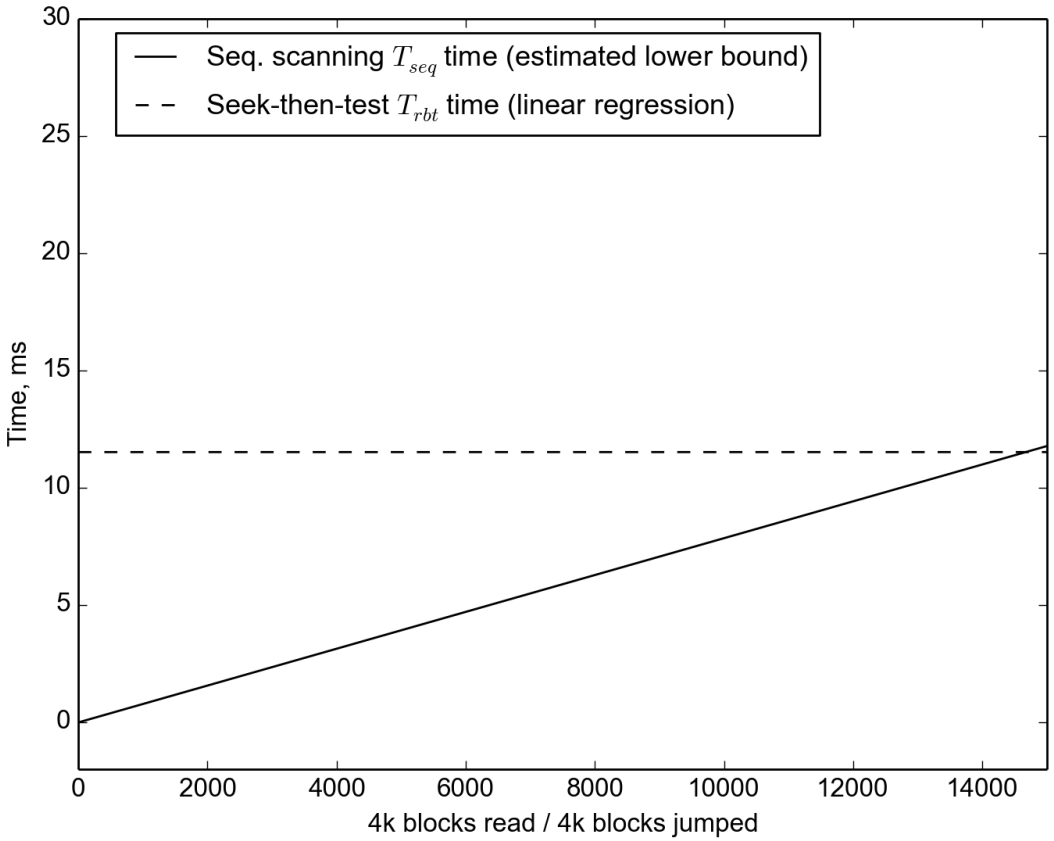

Figure 14: $T_{rbt}(a)$ and $T_{seq}(a)$ calculated for Fujitsu 120 Gb HDD.

files less than 1-2 Mb in size, while for hard disk drives it includes virtually all JPEG files.

Once JPEG data is detected through the block testing, the file carver needs to determine the boundaries of the detected JPEG file. Figure 10 suggests that in absence of drive seek delay, the optimal approach would have been to examine blocks immediately adjacent to the detected block using, for example, signature based JPEG header and footer detection.

In our experiments only the forward directed sequential block reading did not incur drive-seek penalty. The reverse sequential block reading exhibited timing characteristics comparable with random block access. Thus, while sequential carving is optimal for finding JPEG footer, a combination of sequential and random access block reading (e.g. based on binary search) may have to be employed to achieve optimal performance. We note however, that the minimal sizes of high resolution JPEG files are not much bigger than the point of equivalence e between $T_{rbt}(a)$ and $T_{seq}(a)$, and having more that one drive seek operation while searching for JPEG header can be less efficient than sequential carving. For the purposes of this article, we chose to adopt a somewhat simplistic approach that we call *Bounded Sequential Carving*: once our carver detects JPEG data in a sampled block, it performs a single backward jump followed by the usual sequential carving to find the header and footer of the detected drive. We leave analysis of more advanced strategies as a topic for future research.

9. DECA: a decision-theoretic carving program

The analysis given in the preceding sections led us to the file carving algorithm that combines data sampling with sequential carving. It is designed for fast carving of large non-fragmented JPEGs like high resolution pictures produced by DSLR cameras. We named it *DECA* that stands for decision-theoretic carving.

DECA operates on a block device viewed as an array of data blocks B [12]. The user has to specify the minimal size of expected JPEG files l.

DECA alternates between two modes: *data sampling* and *bounded sequential carving*. It starts in the data sampling mode (Algorithm 2). DECA skips the first $l-1$ blocks of B by jumping directly to the block b_{l-1} and reading its content. DECA then runs the Algorithm 1 to check for presence of JPEG data in the block. If no JPEG data is detected, DECA skips the following l blocks and continues sampling at the block b_{2l-1}, but if JPEG data *is* detected at the block b_{l-1}, DECA jumps backwards $l-1$ blocks (to the block b_0) and switches to the bounded sequential carving.

The bounded sequential carving mode (Algorithm 3) is broadly similar to "normal" sequential carving. It reads consecutive data blocks and detects JPEG files using header and footer signatures. The main difference is that it reverts back to the data sampling mode when the JPEG header signature is not found after l blocks of searching

[12] Note that in addition to actual raw block devices, DECA could be applied to partitions, forensic disk images, etc.

Note that DECA does not revert immediately to the sampling mode after finding the JPEG footer. It keeps searching for the next JPEG header for l more blocks. This is important, because humans tend to copy or move JPEG files in groups, as and such JPEG files tend to be stored back-to-back (or within a short distance from each other) on the block device.

Algorithm 2 DECA: Data sampling mode.

INPUT:
array of $m+1$ data blocks $B = (b_0, \cdots, b_m)$;
minimal JPEG file length in blocks l, where $0 \le l \le (m+1)$.
VARIABLES:
index of the next block to be examined pos
contents of the next block to be examined d

```
pos ← (l − 1)
while pos ≤ m do
    JumpTo(b_pos)
    d ← Read(b_pos)
    if JpegDetect(d) then
        pos ← pos − (l − 1)
        pos ← BSC(B, pos, l)        ▷ Bounded seq. carving
        pos ← pos + (l − 1)
    else
        pos ← pos + l
    end if
end while
```

10. Implementation and Evaluation of DECA carver

The previously described DECA algorithm was implemented using C++. The source code can be found in (Gladyshev and James, 2016). The software implementation defaults to linear carving mode. Two of the main functions of the software. First, a disk can be sampled to determine its seek time. With seek-time information, an investigator can determine jump distance at which DECA carving becomes faster than traditional linear carving. Second, the software allows for carving using the DECA algorithm while setting the minimal sector jump distance (in 512 sector increments).

Experimentation seeks to demonstrate the theorized performance of the DECA algorithm, and compare the algorithm to other file carvers commonly used in digital investigations.

10.1. Experimentation Design

DECA is based on estimates of the distribution of JPEG data on a disk. Based on this estimation, DECA attempts to search only the most likely locations for related data. The result of the DECA algorithm is that some sectors will be skipped. In some cases, skipping sectors that are unlikely to contain relevant data may result in carving speed up. However, this method is also likely to miss smaller files.

DECA, in practical terms, can be described simply. If the jump size is high, carve times and number of carved files will decrease. If the jump size is low, carve times will increase and

Algorithm 3 DECA: Bounded sequential carving (BSC).

INPUT:
array of $m+1$ data blocks $B = (b_0, \cdots, b_m)$;
index of the starting data block *start*;
JPEG header search bound (in blocks) l, where $0 \leq l$;
VARIABLES:
index of the next block to be examined pos
contents of the next block to be examined d
index of the first block of JPEG file $jpegpos$
size (in blocks) of the JPEG file *size*
counter of remaining blocks within bound *bound*
CONSTANTS:
maximal allowed size of JPEG file (in blocks) M
RETURNS:
position of the last examined block (pos)

```
pos ← start
bound ← l
JumpTo(b_pos)
while pos ≤ m AND bound > 0 do
    d ← read(b_pos)
    if Header(d) then
        jpegpos ← pos
        size ← 1
        while ¬Footer(d) AND size < M AND pos < m do
            pos ← pos + 1
            size ← size + 1
            d ← read(b_pos)
        end while
        ExtractDataBlocks(b_jpegpos, ··· , b_pos)
        pos ← pos + 1
        bound ← l
    else
        bound ← bound − 1
    end if
end while
return pos
```

the number of carved files will increase. As discussed, this depends on the seek time of the disk holding the data to be analyzed. From the prior description, we propose that DECA is most beneficial when attempting to carve a small number of large files from a disk with a low seek time.

From the description of DECA we propose the following hypotheses for testing:

1. DECA will have a more significant reduction in run-time as the seek time of the disk decreases compared to other carving methods.
2. DECA will have a more significant reduction in run-time compared to other carving methods based on the size of disk/image being analyzed.
3. DECA will normally find fewer files compared to linear carvers.
4. DECA is best suited for carving a small number of large files on a low-seek-time disk.

10.2. Experiment 1

Two disk images were created for testing. One is based on a 4GB USB thumb drive, with a collection of JPEG images along with other documents stored in a FAT32 partition. The second is a 32GB image of a Windows XP operating system containing a user-created documents and JPEG images taken with various digital cameras.

Along with DECA, other carvers tested with the same data sets include Photorec, Scalpel 1.60, Scalpel 2.1 and Foremost. Each of these carvers were configured to only search for JPEG images.

10.3. Evaluation

To test hypothesis 1, each carver was used on the test data set while stored on an sold state drive (SSD) and hard disk drive (HDD). The 4GB image was also tested inside a RAMDisk. RAM cache was cleared before each run. Measurement of carving performance is based on sampling the number of carved files in the carver's output directory (contained on the same disk as the test image) every 0.25 seconds until the carver terminates. All raw test data and testing scripts can be found in the project repository given above.

To determine whether DECA had a reduction in run-time compared to linear carving, we compared the average speed increase over all linear carvers (including DECA in linear carving mode) to the average speed increase between DECA with 7 minimum-block-size configurations (default, 64, 200, 300, 600, 1200, 2400).

The average speed increase for linear carving from HDD to SSD was 237%. The average speed increase for DECA carving from HDD to SSD was 553%. The average speed increase for linear carving from SSD to RAMDisk was 327%. The average speed increase for DECA carving from SSD to RAMDisk was 1229%. Overall, the average speed increase for linear carving was 298%, while the average speed increase for DECA carving was 891%. As expected, when DECA had lower minimum file-size values (64 to 300), performance was equal to or below linear carving on disks with high seek times. Hypothesis 1 is found to be true in most cases except when DECA minimum-file-size is less than 600 on high seek-time disks. The disk seek-time will affect the level of minimum-file-size that should be used.

To test hypothesis 2 we calculated the average speed increase for linear carving and DECA carving from HDD to SSD. For the 4GB image, DECA performed 316% over the linear carver average. For the 32GB image DECA performed 434% over the linear carver average. Similar to before, if DECAs minimum-file-size was below 600, DECA performed at or below the linear carver average for the 32GB drive. As can be seen in Figure 15, DECA saw greater performance increases from HDD to SSD with a larger disk image when minimum-file-size is greater than 600. This is explained by the cumulative effect of continually skipping large sections of the disk.

To test hypothesis 3 we counted the average number of files returned from each carving process. On the 4GB disk image, linear carving returned 356 files on average, and DECA carving returned 30 files on average (regardless of underlying disk).

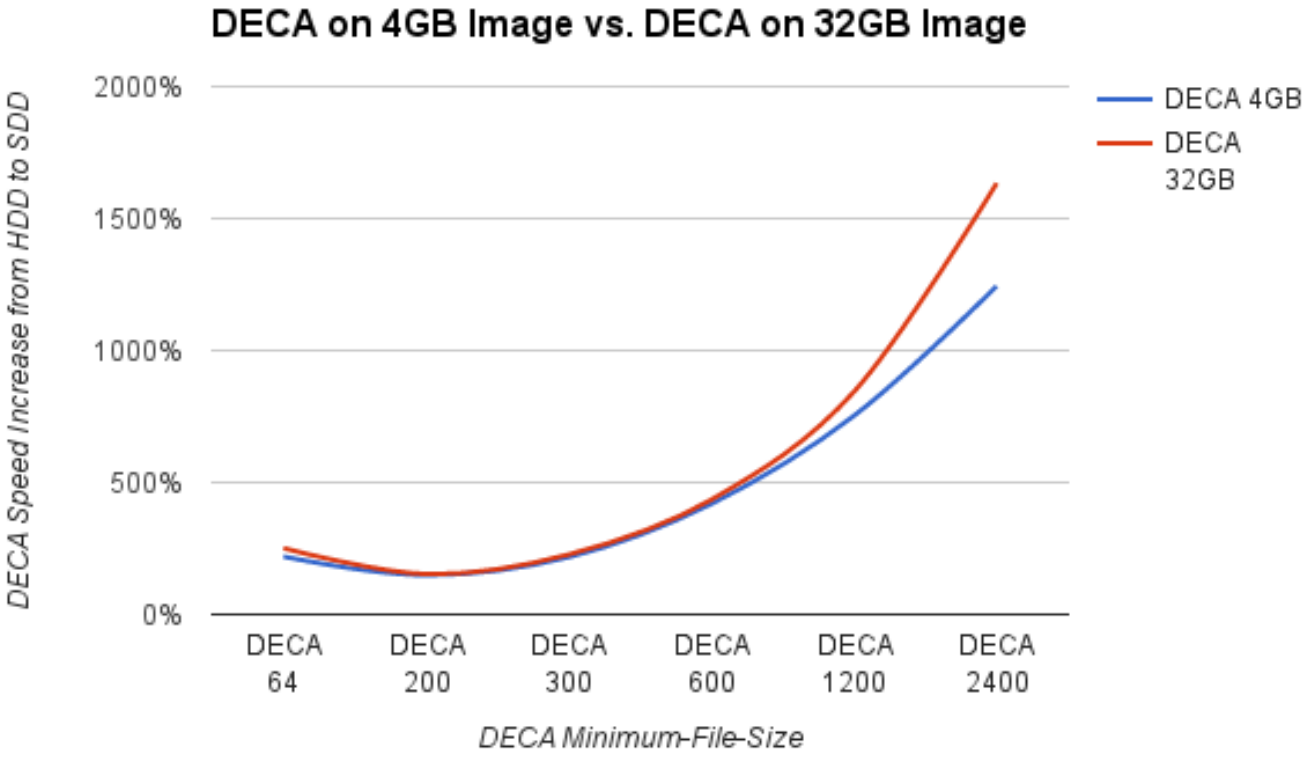

Figure 15: DECA time performance improvements in percentage when comparing carving times on an HDD to carving times on an SSD. This graph shows that DECA performs better with larger disk images when minimum-file-size is greater than 600.

This is consistent with what was predicted since DECA is likely to skip small files and fragments that linear carving will find.

To test hypothesis 4, we examine results of the largest tested DECA minimum-file-size. In our experiments, the largest minimum-file-size used was 2400, which tells DECA that a file is likely to be located within 1,200Kb of its current location. With this setting, DECA was 5 times faster than the linear carving average on an SSD, and 19 times faster than the linear carving average on a RAMDisk. However, DECA only recovered 2% of the files recovered by linear carving. Overall, it appears that hypothesis 3 is also true. By expanding the minimum-file-size DECA accepts and using a disk with a lower seek-time, DECAs speed will dramatically increase at the expensive of smaller file recovery.

10.3.1. Discussion

The DECA algorithm worked as theorized, however, a practical implementation is likely more interesting to investigators. DECA produces significant carving speed increases under certain circumstances, but always at the cost to the number of files carved. Specifically, DECA is ideal for carving a small number of large files across a large, low-seek-time disk. For this reason, DECA may be appropriate for triage purposes, where very fast but incomplete image recovery is acceptable. In cases where the maximum number of files should be carved, or when carving high-seek-time devices (HDD), linear carving largely outperforms DECA.

As discussed, DECA jumps to sections of the disk based on a calculated data distribution. DECA then checks for an image header or footer. If not found, then DECA attempts to classify the data as a JPEG or not. The classifier that was developed could likely be improved. This would increase the number of images correctly recovered, potentially at the cost of speed.

The current implementation of DECA focuses on JPEG images, while imagining current digital cameras that produce images greater than 1 megabyte. However, this algorithm may be suitable for other specific file types, such as music, video, zip files and email containers. This method could also be used in conjunction with other carving methods, such as in-place carving (Richard et al., 2007; Meijer, Rob, 2012).

11. Conclusions

Decision theoretic analysis allows a file carver to consider the most likely locations of relevant data based on what is currently known about the distribution of data on the disk. By skipping sections of the disk that are unlikely to contain relevant data, carving times can be greatly decreased, where speed benefits are cumulative as the disk becomes larger. Through a practical implementation we demonstrated that decision theoretic analysis is useful when carving a small number of files from a large, low-seek-time disk. With current storage disk technology, decision theoretic carving (DECA) is best implemented as a triage solution. For traditional carving needs, linear carving gives more comprehensive results, and on standard hard disk drives, also has similar carving speeds.

11.1. Future Work

This work is a first effort to propose decision theoretic analysis applied to file carving. There are currently a number of limitations to the practical implementation (DECA) carver. Specifically, file distribution has been pre-calculated based on surveyed data in other drives. Future work will attempt to find a method for calculating a data distribution on the disk to be analyzed without increasing carving time significantly. Next, as DECA jumps to a location on the disk it must detect whether the data stored in that location is relevant. We have attempted several detection methods, most of which significantly increased the overall run time. We will seek to improve the data classifier that DECA uses.

References

Casey, E., Ferraro, M., Nguyen, L., nov 2009. Investigation Delayed Is Justice Denied: Proposals for Expediting Forensic Examinations of Digital Evidence*. Journal of Forensic Sciences 54 (6), 1353–1364.

Garfinkel, S. L., 2007. Carving contiguous and fragmented files with fast object validation. digital investigation 4, 2–12.

Garfinkel, S. L., McCarrin, M., aug 2015. Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb. Digital Investigation 14, S95–S105.

Gladyshev, P., James, J. I., 2016. Deca source code.
URL `https://bitbucket.org/pavel_gladyshev/deca-dij.git`

Grenier, C., 2007. Photorec.
URL `http://www.cgsecurity.org/wiki/PhotoRec`

James, J. I., Lopez-Fernandez, A., Gladyhsev, P., 2014. Measuring accuracy of automated parsing and categorization tools and processes in digital investigations. In: Digital Forensics and Cyber Crime. Springer, pp. 147–169.

Jaynes, E. T., 2003. Probability theory: the logic of science. Cambridge university press.

Koopmans, M. B., James, J. I., sep 2013. Automated network triage. Digital Investigation 10 (2), 129–137.

Li, Q., Ong, A., Suganthan, P., Thing, V., 2011. A novel support vector machine approach to high entropy data fragment classification. SAISMC, 236–247.

Marturana, F., Tacconi, S., 2013. A machine learning-based triage methodology for automated categorization of digital media. Digital Investigation 10 (2), 193–204.

Meijer, Rob, may 2012. The carve path zero-storage library and filesystem.
URL `https://github.com/DNPA/carvfs`

Memon, N., Pal, A., 2006. Automated reassembly of file fragmented images using greedy algorithms. Image Processing, IEEE Transactions on 15 (2), 385–393.

Overill, R. E., Silomon, J. A., Roscoe, K. A., 2013. Triage template pipelines in digital forensic investigations. Digital Investigation 10 (2), 168–174.

Pollitt, M. M., 2013. Triage: A practical solution or admission of failure. Digital Investigation 10 (2), 87–88.

Richard, G., Roussev, V., Marziale, L., 2007. In-Place File Carving. In: Craiger, P., Shenoi, S. (Eds.), Advances in Digital Forensics III. Vol. 242. Springer New York, New York, NY, pp. 217–230.

Richard III, G. G., Roussev, V., 2005. Scalpel: A frugal, high performance file carver. In: DFRWS.

Rogers, M., 2003. The role of criminal profiling in the computer forensics process. Computers & Security 22 (4), 292–298.

Schell, B. H., Martin, M. V., Hung, P. C., Rueda, L., 2007. Cyber child pornography: A review paper of the social and legal issues and remediesand a proposed technological solution. Aggression and violent behavior 12 (1), 45–63.

Shaw, A., Browne, A., 2013. A practical and robust approach to coping with large volumes of data submitted for digital forensic examination. Digital Investigation 10 (2), 116–128.

Veenman, C. J., aug 2007. Statistical Disk Cluster Classification for File Carving. In: Third International Symposium on Information Assurance and Security. IEEE, pp. 393–398.