Detecting File Fragmentation Point Using Sequential Hypothesis Testing

By

Anandabrata Pal, Husrev Sencar, Nasir Memon

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2008 USA

Baltimore, MD (Aug 11th - 13th)

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Detecting file fragmentation point using sequential hypothesis testing

Anandabrata Pal[*], *Husrev T. Sencar*, *Nasir Memon*

Computer Science Department, Polytechnic University, 6 Metrotech Center, Brooklyn, NY 11201, United States

Keywords:
File carving
Data recovery
Forensics
Fragmentation, Sequential hypothesis testing
DFRWS carving challenge

A B S T R A C T

File carving is a technique whereby data files are extracted from a digital device without the assistance of file tables or other disk meta-data. One of the primary challenges in file carving can be found in attempting to recover files that are fragmented. In this paper, we show how detecting the point of fragmentation of a file can benefit fragmented file recovery. We then present a sequential hypothesis testing procedure to identify the fragmentation point of a file by sequentially comparing adjacent pairs of blocks from the starting block of a file until the fragmentation point is reached. By utilizing serial analysis we are able to minimize the errors in detecting the fragmentation points. The performance results obtained from the fragmented test-sets of DFRWS 2006 and 2007 show that the method can be effectively used in recovery of fragmented files.

1. Introduction

With the ever increasing adoption of digital storage mediums for both legitimate and criminal use, the need for more sophisticated data recovery and forensic recovery products has also increased. Most file systems and storage devices store data by dividing it into many clusters and by maintaining the list of clusters (file table) used for storing each file's data.[1] When a file is accessed the data is retrieved in sequence from this list of clusters. Similarly, deletion of a file is typically realized by removing a file's entry from the file table. Traditional data recovery and forensics products attempt to recover data by analyzing the file system and extracting the data pointed to by the file system. Traditional recovery techniques fail to recover data when the file system is corrupted, not present or has missing entries. File carving was then introduced to recover files from the "unallocated" space of a disk, i.e., the area of the disk not pointed to by the file system. The initial and still by far most common form of file carvers simply analyze headers and footers of a file and attempt to merge all the blocks in between. One of the most well known of these file carvers is Scalpel (Richard and Roussev, 2005). However, these file carvers still fail to recover files that are fragmented.

A file is said to be fragmented when it is not stored on a continuum of clusters, and the most difficult challenge in data carving is to recover files when they are fragmented into two or more pieces. Garfinkel (2007) determined that fragmentation on a typical disk is less than 10%, however, the fragmentation level of forensically important file types (like images, office files, and email) is relatively high. Among his main findings are that up to 42% of PST files (outlook email) 17% of MS-Word files and 16% of JPEGs are fragmented. It is, therefore, clear that recovery of fragmented files is a critical problem in forensics.

While the starting and end points of a file can be identified by specific markers (e.g., file headers and footers), the point at which a file fragments and the point at which the next fragment starts can be extremely difficult to ascertain. Identifying

* *Corresponding author*. Tel.: +1 917 482 0211.
E-mail address: PashaPal@hotmail.com (A. Pal).

[1] For example, in the file system FAT-32 the root table entry with the file name will point to the first cluster of the file, which in turn will point to the next cluster and so on until the last cluster of the file.

1742-2876/$ – see front matter © 2008 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2008.05.015

these points involves a detailed understanding of individual file formats, and existing techniques fail to scale when dealing with hundreds of thousands of blocks, and files that may be fragmented into more than two fragments.

In this paper, we show how by identifying the fragmentation point of a file we can improve the performance of Garfinkel's (2007) bifragment gap carving technique, as well as Pal and Memon's (2006) Parallel Unique Path (PUP) techniques for recovering fragmented files. We present a technique to identify the fragmentation point(s) of a file by utilizing sequential hypothesis test (SHT) procedure. The technique begins with a header block identifying the start of a file and then attempts to validate via SHT each subsequent block following the header block. The fragmentation point is identified when SHT identifies a block as not belonging to the file. By utilizing this technique, we are able to correctly and efficiently recover JPEG images from the DFRWS 2006 (Carrier et al., 2006) and 2007 (Carrier et al., 2007) test-sets even in the presence of tens of thousands of blocks and files fragmented into three or more parts. The bifragment gap carving technique enhanced with SHT allows us to improve the performance result of DFRWS 2006 challenge test-sets, although the technique cannot be used for DFRWS 2007. We then show how Parallel Unique Path enhanced with SHT is able to recover all fragmented JPEGs from DFRWS 2006 and all recoverable JPEGs from 2007 challenge test-sets. As far as we are aware, no other automated technique can recover multi-fragmented JPEGs from the DFRWS 2007 test set.

The next section begins with a description of fragmentation and how fragmentation occurs on disks. We then define the basic terms used throughout the paper. Section 3 describes existing techniques for fragmented file recovery, followed by Section 4 which describes our technique for fragmentation point detection. We then formalize the fragmentation point detection problem and describe our solution in Section 5. Section 6 contains information about our refined file carving system using Fragmentation Point Detection based on the PUP carver. We detail our experiments and results in Section 7 for the new file carver as well as bifragment gap carving. We conclude with some open problems as well as areas that we are currently looking into.

2. Fragmentation

File fragmentation is said to occur when a file is not stored in the correct sequence on consecutive blocks on disk. In other words if a file is fragmented, the sequence of blocks from the start of a file to the end of the file will result in an incorrect reconstruction of the file. Fig. 1 provides a simplified example of a fragmented file. In the figure, the file J_1 has been broken into two fragments. The first fragment starts at block 1 and ends at block 4. The second fragment starts at block 8 and ends at block 9. This file is considered to be bi-fragmented as it has only two fragments. Garfinkel (2007) showed that bi-fragmented fragmentation is the most common type of fragmentation, however, files fragmented into three or more pieces are not uncommon.

Garfinkel's fragmentation statistics come from identifying over 350 disks containing FAT, NTFS and UFS file systems.

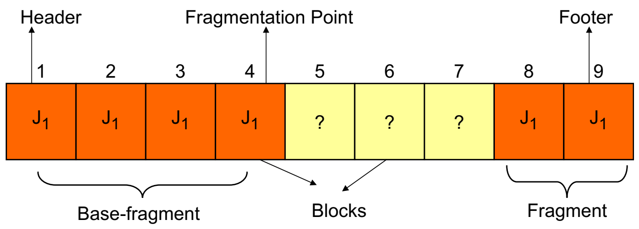

Fig. 1 – File J1 has been broken into two fragments spanning six blocks, with three blocks in between not belonging to J1.

Fragmentation typically occurs under one of the following scenarios:

(1) *Low disk space:* If the disk space is low and the disk is not defragmented, there may be many small groups of blocks/clusters that are available for storing information. However, future files to be stored may be larger than the largest of these free groups of blocks, and as a result a file may need to be fragmented across multiple of these blocks.
(2) *Appending/editing files:* If a file is saved on disk and then additional files are also saved starting at the cluster that the original file ended at, then fragmentation may occur if the original file is then appended to (and increases in size larger than the cluster size). Some file systems like the Amiga Smart Filesystem may attempt to move the whole file in such scenarios. Some other file systems like UFS attempt to provide "extents" which are attempts to pre-allocate longer chunks in anticipation of appending (McVoy and Kleiman, 1991). Another technique called delayed allocation used in file systems like XFS (Sweeney et al., 1996) and ZFS reserve file system blocks but attempt to delay the physical allocation of the blocks until the operating system forces a flushing of the contents. However, while some of these techniques are able to reduce fragmentation they are unable to eliminate fragmentation completely.
(3) *Wear-leveling algorithms in next generation devices:* Solid State Devices are currently utilizing proprietary wear-leveling algorithms to store data on the disk (STORAGEsearch.com). If the information mapping the logical geometry of the disk to the physical geometry is destroyed or gets corrupted, then any data extracted will be fragmented, with no easy way of determining the correct sequence of blocks to recover files.
(4) *File system:* In rare cases the file system itself will force fragmentation. The Unix File System will fragment files that are long or have bytes at the end of the file that will not fit into an even number of sectors (Carrier, 2005).

In Table 1, we give basic definitions concerning fragmentation that will be used throughout the paper and Fig. 1 provides a simple visualization of these definitions.

As mentioned earlier, once a file has been fragmented, traditional file carving techniques will fail to recover the file. In the next section, we detail the process required to recover fragmented files and provide a description of existing techniques to achieve this.

Table 1 – Definitions for terms used in paper

Term	Definition
Block	This is the size of the smallest data unit that can be written to disk which can be either a disk sector or cluster. To avoid confusion we will use the term *block* and b_y will denote the block numbered y in the access order.
Header	This is a block that contains the starting point of a file.
Footer	This is a block that contains the ending data of a file.
Fragment	A fragment is considered to be one or more blocks of a file that are not sequentially connected to other blocks of the same file. Fragmented files are considered to have two or more fragments (though one or more of these may not be present on the disk anymore). Each fragment of a file is assumed to be separated from each other by an unknown number of blocks.
Base-fragment	The starting fragment of a file that contains the header as its first block.
Fragmentation point	This is the last block belonging to a file before fragmentation occurs. A file may have multiple fragmentation points if it has multiple fragments.
Fragmentation area	A set of consecutive blocks b_y, b_{y+1}, b_{y+2}, $b_{y+3}\ldots$ containing the fragmentation point.

3. Fragmented file carving

To recover fragmented files correctly a file carver must be able to determine the starting point of a file and the correct blocks that are required to reconstruct the file. In essence it is a three step process:

(1) Identify starting point of a file.
(2) Identify blocks belonging to file.
(3) Order the blocks correctly to reconstruct the file.

There are two published techniques that attempt to follow this three step process in differing ways. We now describe these two techniques.

3.1. *Bifragment gap carving*

Garfinkel (2007) introduced the *fast object validation* technique for recovery of fragmented files. This technique recovers files that have headers and footers and are fragmented into two fragments (bi-fragmented). This technique works only for files that can be validated/decoded. Decoding is the process of transforming information in the data blocks associated with a file into its original format that describes the actual content. Many file types, like JPEG, MPEG, ZIP, etc., have to be decoded before their content can be understood. Object validation is the process of verifying if a file obeys the structured rules of its file type. Therefore, an object validator will indicate whether a block violates the structure or rules required of the specific file or file type. For example in the PNG file format, the data can be validated through cyclic redundancy checking, and a mismatch will indicate either data corruption or fragmentation. A decoder can be trivially used as an object validator by observing whether or not it can correctly decode each block in the sequence.

Bifragment Gap Carving (BGC) recovery occurs by exhaustively searching all combinations of blocks between an identified header and footer while excluding different number of blocks until a successful decoding/validation is possible. We now describe bifragment gap carving in greater detail. Let b_h be the header block, b_f be the last block of the first fragment of a file, b_s be the starting block of the second fragment, and b_z be the footer block. Blocks b_h and b_z are known and b_f and b_s have to be determined. For each gap size g, starting with size one, all combinations of b_f and b_s are designated so that they are exactly g blocks apart, i.e., $s-f=g$. A validator/decoder is then run on the byte stream representing blocks b_h to b_f and b_s to b_z. If the validation fails b_f and b_s are readjusted and the process continued until all choices of b_f and b_s are tried for that gap size, after which the gap size is incremented. This continues until a validation returns true or the gap size can't be increased any further. This technique performs satisfactorily when the two fragments are close to each other; however, it has the following limitations for the more general case.

(1) The technique does not scale for files fragmented with large gaps. If n is the number of blocks between b_h and b_z then in the worst case n^2 object validations may be required before a successful recovery.
(2) For files with more than two fragments, the number of object validations that need to be performed can be impractically high. This is because $n-1$ gap sizes have to be used in parallel where n is the number of fragments in the file. It is also very highly unlikely that n can be determined before hand.
(3) Successful decoding/validation does not always imply that a file was reconstructed correctly. Decoders will give an error when the data in the blocks do not conform to inherent decoding rules or structure. For example, standard JPEG decoder will stop with an error when a retrieved bit pattern has no corresponding entry in the Huffman code table. Fig. 2 from DFRWS 2007 is an example of a successfully decoded but incorrect JPEG file.
(4) Many file types cannot be validated by their structure or do not require decoding. For example, 24-bit BMPs have a header, followed by pixel values where each pixel is represented with three bytes, and has no footer. If a bitmap image is fragmented, any block can be considered to be a candidate for the fragmentation point and the starting point of another fragment. Object validation will fail in such a case.
(5) Missing or corrupted blocks for a file will result in the worst case often.

3.2. *Graph theoretic carving*

Pal and Memon (2006) formulate the image reassembly problem as a k-vertex disjoint graph problem and reassembly is then done by finding an optimal ordering of blocks. Their

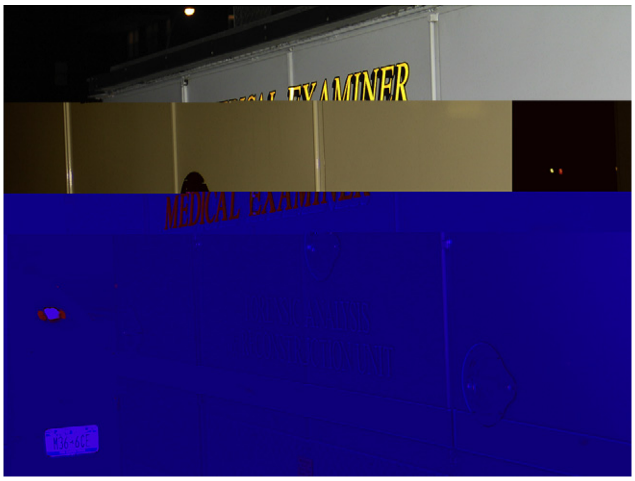

Fig. 2 – Fully validated but incorrect JPEG image from the DFRWS 2007 test set.

technique does not require object validations or decoding, but utilizes a matching metric to indicate the likelihood that a block follows another. For file types that can't be validated based on their structure (24-bit BMPs), analysis of the actual contents of each file is required to determine if a block should be paired with another block. However, even if a block can be validated the contents of each block are still analyzed and matching metrics created. Matching metrics differ according to the file type. For example in images, the matching metric is generated by analyzing the pixel boundary created by the merging of two blocks.

Utilizing the matching metric, they present three algorithms using two heuristics. For the purpose of this paper we describe the Parallel Unique Path (PUP) algorithm using the greedy heuristic as this has the best combination of performance and results. PUP is a modified Dijkstra's (1959) single source shortest path algorithm, used to reassemble multiple files simultaneously. Starting with the file headers of each file, the best match for each header is chosen and then the header–block pair with the best of all the best matches is merged to the header. The process is repeated until all files are reconstructed.

More formally, the k file headers (b_{h1}, b_{h2}, … b_{hk}) are stored as the starting blocks in the reconstruction paths P_i for each of the k files. A set $S = (b_{s1}, b_{s2}, \ldots, b_{sk})$ of current blocks is maintained for processing, where b_{si} is the current block for the ith file. Initially, all the k starting header blocks are stored as the current blocks for each file (i.e., $b_{si} = b_{hi}$). The best greedy match for each of the k starting blocks is then found and stored in the set $T = (b_{t1}, b_{t2}, \ldots, b_{tk})$ where b_{ti} represents the best match for b_{si}. From the set T of best matches the block with the overall best matching metric is chosen.

Assuming that this best block is b_{ti}, the following steps are undertaken:

(1) Add b_{ti} to reconstruction path of ith file, (i.e., $P_i = P_i \| b_{ti}$).
(2) Replace current block in set S for ith file (i.e., $b_{si} = b_{hi}$).
(3) Evaluate new set T of best matches for S.
(4) Again find best block b_{ti} in T.
(5) Repeat 1 until all files are built.

Fig. 3 shows an example of the algorithm where there are three files being reconstructed. Fig. 3(a) shows the header blocks H_1, H_2 and H_3 of the three files and their best matches. The best of all the matches is presented with a dotted line and is the H_2–6 pair of blocks. Fig. 3(b) now shows the new set of best matches after block 6 has been added to the reconstruction path of H_2. Now block 4 is chosen once each for blocks H_1 and 6. However, the pair H_1–4 is the best and therefore 4 is added to the reconstruction path of H_1 and the next best match for block 6 is determined Fig. 3(c). This process continues until all files are reconstructed.

While the reported results are very promising, the reassembly requires O(n^2) computations, where n is the total number of blocks. Clearly, this system fails to scale when dealing with tens of thousands and even millions of blocks. The reason O(n^2) computations are necessary is due to the assumption that fragmentation occurs completely randomly and fragment sizes can be as small as a single block. However, since it assumes random fragmentation, it is able to handle files fragmented into greater than two fragments. Another problem with this technique is that it assumes all blocks are present and that there are no holes.

3.3. *The need for fragmentation point detection*

As mentioned both BGC and PUP have problems when dealing with fragmented files. Both require O(n^2) computations in the worst case and fail to scale for very large gaps or a file fragmented into many pieces. PUP assumes that fragmentation is completely random, however, as shown in Garfinkel (2007), files with greater than three fragments are very rare, and file systems will almost never fragment a file on a block by block basis. BGC assumes if a file validates it is correct, and does not attempt to validate or score individual pairs of blocks.

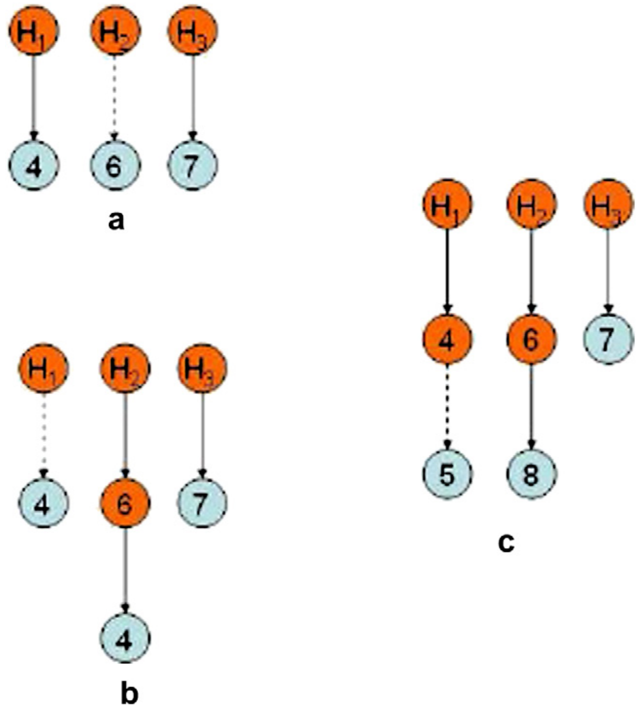

Fig. 3 – Simplified example of PUP algorithm.

Both techniques would benefit from being able to identify the base-fragment, which as defined earlier, is the first fragment of a file. More specifically, the last block of the base-fragment (i.e., fragmentation point) needs to be detected. In the case of BGC, correct fragmentation point identification allows the assumed fragmentation ending point b_f (as defined earlier) to be fixed, and only b_s (next fragment starting point) adjusted, thus greatly reducing the number of computations required to validate a file. Even if the fragmentation point was not found, but the fragmentation area was found (a small set of blocks, one of which contains the fragmentation point), b_f would only need to be adjusted within the fragmentation area and not from the header. In the case of PUP rather than assuming each block is randomly fragmented, a modification can be made so that if a block is chosen to belong to a file, then the blocks following the chosen one are sequentially compared to the previous block, until a fragmentation point is reached, at which point the algorithm will resume as normal. In the next section we present some simple techniques that can be used for fragmentation point detection.

4. Fragmentation point detection

In Fig. 4, a JPEG image J_1 has one file fragmentation point at the fourth block, b_4. This implies that blocks 1–4 all belong together in order. If a file is fragmented into more than two pieces, then multiple file fragmentation points will exist, and if the file has no fragmentation then there will be no file fragmentation points. For a file fragmented into more than two pieces, the techniques for identifying the starting file fragmentation point are no different than the techniques for identifying subsequent file fragmentation points.

Other than the rare scenario where a file is fragmented because two or more of its fragments are swapped with each other, the gap between each of a file's fragment ending points and the next correct fragment's starting point contains data that does not belong to the file. The following tests utilize this information to identify the fragmentation point.

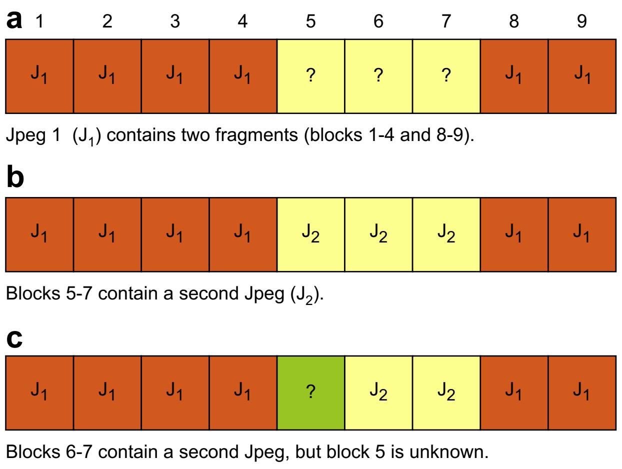
a
1 2 3 4 5 6 7 8 9
J1 J1 J1 J1 ? ? ? J1 J1
Jpeg 1 (J1) contains two fragments (blocks 1-4 and 8-9).
b
J1 J1 J1 J1 J2 J2 J2 J1 J1
Blocks 5-7 contain a second Jpeg (J2).
c
J1 J1 J1 J1 ? J2 J2 J1 J1
Blocks 6-7 contain a second Jpeg, but block 5 is unknown.

Fig. 4 – Three examples of different types of blocks in between the two fragments of a JPEG J_1.

4.1. *Syntactical tests*

With this approach the fragmentation point is detected by identifying whether or not a block belongs to a file in question through one of the following methods:

- *Using keywords and signatures indicating different file types.* During recovery of a fragmented file, if a block is found to belong to some other file or file type, then it is assumed that a fragmentation point is reached. For example, while recovering a JPEG file if a block is identified to be starting with an HTML file header and has a few other HTML tags, then the fragmentation point is deemed to be detected. Similarly, certain keywords are not expected to be seen during the process of recovery (like invalid JPEG markers), and they can be used for detecting fragmentation points.
- *Content analysis indicating incorrect block.* An abrupt change in characteristics of the data might potentially indicate a fragmentation point. For example during the recovery of a JPEG file if one were to encounter blocks containing English words, this would indicate that the fragmentation has occurred.

With this type of approach while we may say with certainty that a block does not belong to a particular file, these methods on their own have no way of determining whether or not the previous blocks belong to the file. In the simple example shown in Fig. 4, a JPEG file has been fragmented into two pieces, the first piece, numbering four blocks, and the last piece, numbering three blocks, with three unknown blocks in between. If the first block has the starting (header) signature of another JPEG as shown in Fig. 4b, then this may yield to the decision that fragmentation happened at the fifth block and the last correct block was the fourth block. Moreover, when multiple files of the same type are fragmented together with this approach it will be much harder to detect the presence of fragmentation.

4.2. *Statistical tests*

Statistical tests attempt to compare the statistics of each block to a model for each file type and then classify the block. Some examples of statistical tests involve entropy of each block and the OSCAR method (Karresand and Shahmehri, 2006a,b). The Oscar method is based on building models, called centroids, of the mean and standard deviation of the byte frequency distribution of different file types. A block is compared to all models and a determination made as to which file type it seems to conform to. Again these tests suffer from the same problems of being unable to detect the actual fragmentation point that the syntactical tests suffer from, but in addition, blocks can be falsely identified as belonging to another file type.

4.3. *Basic sequential validation*

Another simple technique to identify the fragmentation point is to start validating blocks sequentially from the header and continuing on until the validator (decoder) stops with an error. With this technique the last correctly validated block is deemed to be the fragmentation point. In Fig. 4c validation starts at the first block and continues until it fails at block five, leading to the conclusion that fragmentation point is at block 4.

However, it is possible for a validator to successfully validate random blocks of data, which will result in an inaccurate recovery of a file. In fact, this is quite common and is not at all unusual. Take a look at Fig. 5, this shows four images from DFRWS 2007 that were decoded and recovered incorrectly. In DFRWS 2007 sequential decoding alone will result in 8 of 18 JPEGs having the fragmentation point identified incorrectly because multiple blocks beyond the correct fragmentation point will be validated via decoding.

5. Fragmentation point detection using sequential hypothesis testing

The main focus of this paper is to improve on PUP and BGC recovery techniques by assuming a more realistic fragmentation scenario where fragments are not randomly scattered but have multiple blocks sequentially stored. However, at the same time we do not want to rely on basic decoding/validation techniques alone to determine where a fragmentation point may occur. We begin by reliably and efficiently detecting the fragmentation point b_f and then attempt to find b_s, the starting block of the next fragment. For this purpose, we propose a general fragmentation point detection method which is then utilized as a part of a JPEG image file recovery method.

5.1. *Problem formulation*

Recall that we define a *base-fragment* to be the first fragment of a file. It is composed of k physical data blocks that start with an identifiable bit pattern or a header. Our objective is to determine the total number of blocks k within a *base-fragment* while minimizing decision errors in falsely identifying the fragmentation point. A decision error can be in two forms:

- a random block that does not belong to the actual fragment is joined, i.e., *false addition*; or
- a block is separated from the fragment that it belongs to, i.e., *false elimination*.

Hence, given the beginning of a *base-fragment*, a binary decision is made for each subsequent data block to determine as to whether or not a given data block belongs to the *base-fragment*. This problem can be formulated as a hypothesis test and the corresponding analysis' framework, based on false-positive and false-detection probabilities, can be extended to false-addition and false elimination probabilities.

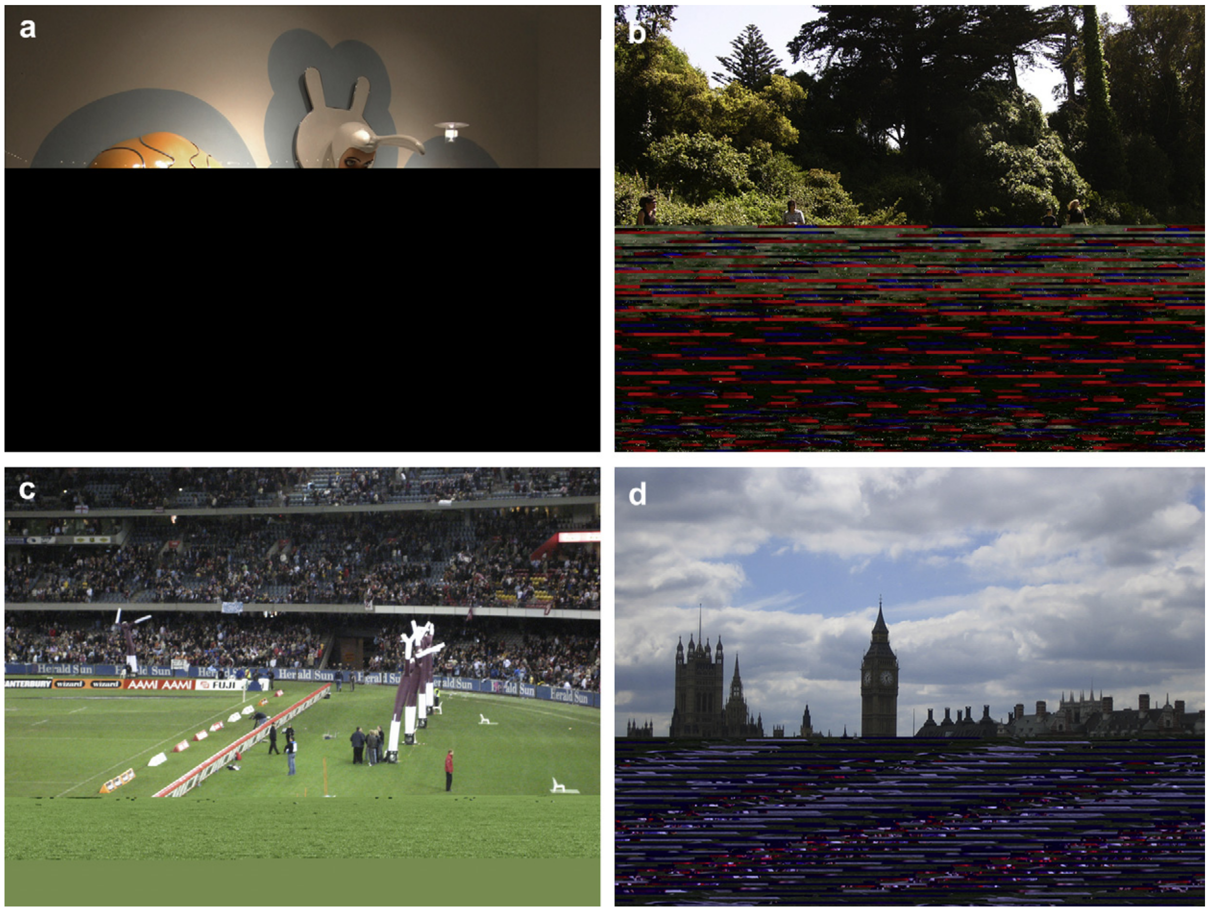

Fig. 5 – Four images from the DFRWS 2007 test-set that are decoded incorrectly when doing sequential decoding.

For this purpose, we define a *matching metric* Y_i that may be computed between the data block b_i in question and already identified parts of the *base-fragment*, i.e., $b_0, b_1, \ldots, b_{i-1}$.

In a conventional hypothesis test, for each data block a matching metric is computed and an individual hard decision is made for each block in the order they appear on the storage medium until a fragmentation point is detected. However, since data blocks before the fragmentation point are more likely to be appended to the base-fragment than blocks beyond the fragmentation point, decisions made based on a sequence of blocks, as opposed to a single block, will be more reliable. It should also be noted that in fragmentation point detection, false addition of a random block is typically much more costly than a false elimination of a correct block as those data blocks can be later correctly merged in the next rounds where all the un-attributed data blocks will be matched against successfully reconstructed base-fragments. On the other hand, falsely merged data blocks will not only cause an error in reconstruction of a base-fragment but they may also curtail the correct reconstruction of other fragments (and files) due to loss of continuity.

5.2. *Forward fragment point detection test*

In our sequential fragment detection method, the number of observations Y_1, Y_2, Y_3, …, associated with a set of blocks, is not fixed in advance. Instead, they are evaluated collectively as they become available and the test is ended in favor of a decision only when the resulting decision statistic is significantly low or high. Otherwise, if the statistic is in between these two bounds, the test is continued. Starting with the first data block of the base-fragment b_0, identified by a fixed pattern or header, subsequent data blocks b_1, b_2, …b_n are appended to b_0 and a matching metric, $Y_1, Y_2, \ldots, Y_n$ is obtained in sequence with each addition. Accordingly, we define the hypotheses H_0 and H_1 as following.

H_0: blocks $b_1, b_2, \ldots, b_n$ belong in sequence to fragment;
H_1: blocks $b_1, b_2, \ldots, b_n$ do not belong in sequence to fragment.

If the evaluated data blocks $b_1, b_2, \ldots, b_n$ do not yield to a conclusive decision, the test continues with the inclusion of block b_{n+1} until one of the hypotheses is confirmed. When hypothesis H_0 is true, the evaluated blocks are merged to the base-fragment and a new test is started. Each time the test starts with a new data block in sequence, the matching statistic is computed with respect to the recovered part of the base-fragment. The test procedure finalizes after one of the following conditions occur:

(1) H_1 is achieved.
(2) File is completely recovered.
(3) An error occurs because no data-block remains or remaining blocks are of a different file type.

Let $\mathbf{Y}$ represent the sequence $Y_1, Y_2, \ldots, Y_n$, then in a sequential hypothesis test, a test statistic Λ is computed as the likelihood ratio of observing sequence $\mathbf{Y}$ under the two hypotheses, which is expressed as the ratio of the conditional distributions of observed matching metrics under H_0 and H_1 as

$$\Lambda(\mathbf{Y}) = \frac{\Pr(\mathbf{Y}|H_1)}{\Pr(\mathbf{Y}|H_0)}. \tag{1}$$

Finally, a decision is made by comparing Λ to appropriately selected thresholds as

$$\text{outcome of test} = \begin{cases} H_1, & \Lambda(Y) > \tau^+ \\ H_0, & \Lambda(Y) < \tau_- \\ \text{inconclusive}, & \tau^- < \Lambda(Y) < \tau^+ \end{cases} \tag{2}$$

That is, if the test statistic (likelihood ratio) is larger than τ^+, we assume that hypothesis H_1 is true and we have found the fragmentation region. If, on the other hand, test statistic is smaller than τ^-, hypothesis H_0 is true and all the fragments are merged and the test continues from the next sequential block. Finally, if neither cases are true testing continues until one of the thresholds is exceeded or end-of-file indicator is reached. The thresholds τ^- and τ^+ can be chosen so as to upper bound the probability of errors due to false-eliminations, P_{fr}, and false additions, P_{fe} as (Wald, 1947)

$$\tau^+ = \frac{1 - P_{fe}}{P_{fa}} \text{ and } \tau^- = \frac{P_{fe}}{1 - P_{fa}}. \tag{3}$$

Ultimately, the success of the sequential fragment point detection method depends on two factors. The first factor is the choice of the matching-metric whose design has to take into consideration different file types and to capture semantic or syntactic characteristics of the file. The second factor is the accurate determination of the conditional probability mass functions under the two hypotheses. This essentially boils down to obtaining multi-dimensional probability distribution functions under two hypotheses. This requires access to statistics revealing typical number of data blocks in a file fragment which might depend on various parameters like disk size, operating system, disk usage activity, etc. Lacking this information, we assume that the matching-metrics computed from each sequentially stored block are independent and identically distributed (iid). It should be noted that when the data-block locations are not necessarily sequential, iid assumption holds. For example, when all the base-fragments are recovered and subsequent fragment for each file have to be identified, from among all the remaining un-associated data blocks, such a model fits very well. Under iid assumption, the test statistic can be rewritten as

$$\Lambda(Y) \equiv \frac{\Pr(\mathbf{Y}|H_1)}{\Pr(\mathbf{Y}|H_0)} = \prod_{i=1}^{n} \frac{\Pr(Y_i|H_1)}{\Pr(Y_i|H_0)}. \tag{4}$$

In this case, the test statistic can be incrementally updated with each observation through multiplication by probability ratio associated with the most recently considered data-block. The test statistic is later compared to two thresholds to confirm a hypothesis or to continue the test.

5.3. *Reverse fragmentation point detection*

As noted earlier, falsely merging a data-block with or eliminating it from a base-fragment will cause an error in the recovery process. Therefore, our main goal is to determine the fragment boundary by identifying the last data block in sequence that belongs to the base-fragment. Although the above sequential fragment detection procedure provides an advantage in avoiding false additions and false-eliminations by not making a decision when the information is inconclusive, it also makes accurate fragmentation point detection difficult. This is because each decision is made only after a window of observations is made, and a fragmentation point can be anywhere in this sequence and the test cannot accurately determine it.

To cope with this problem we propose to conduct a reverse sequential test every time the (forward) test is inconclusive and the test statistic has to be updated by considering a new data-block. Since for a given window of observations (in a forward test) $Y_1, Y_2, \ldots, Y_n$ the observations towards the end of the block are more likely to be due to random data blocks, by visiting them first reverse test enables more accurate detection of the fragmentation point. In a reverse hypothesis test, the test statistic is computed by traversing the measured matching-metrics in reverse order, i.e., sequence $Y_n, Y_{n-1}, \ldots, Y_2$. The test starts by considering Y_n only. If the test evaluates H_1 as true (b_n does not belong to base-fragment), the test is started with Y_{n-1}, and if H_1 is evaluated true the reverse test is terminated and the forward testing continues. On the other hand, if the reverse test is inconclusive, the test continues by considering the next to last data blocks, i.e., Y_n, Y_{n-1}, until a decision is reached. The fragmentation point is deemed to be the most recently eliminated data-block before the test finally confirms the H_1 hypothesis. Fig. 6 shows the flow diagram of the fragment point detection algorithm.

Another issue is that the recovery of the base-fragment will be continued via merging new blocks to it every time the test terminates in favor of H_0. However, it must be noted that the designated test statistics Y, by its definition, is computed between the known parts of the base-fragment and the blocks in question. Therefore, when both the forward and backward tests are inconclusive and further blocks considered do not form a boundary with the recovered part of the image, the test has to be forcefully terminated. In this case the fragmentation point is assigned to the last block that ended the test in favor of the H_0 hypothesis and the blocks that cannot be decided are not added to the base-fragment.

Having identified the fragmentation point by itself is not enough (unless the fragmentation point is the end of file) for recovery. Once the base-fragment has been identified, the starting block of the next fragment belonging to the file being reconstructed needs to be determined. In the next section we present a new model for a file carver and then present a modified version of the PUP algorithm that utilizes SHT for a complete recovery solution.

It should be remembered that false eliminations (fragmentation point is identified before actual identification point) are preferable over false additions (late detection of fragmentation point), since the second step of the carving (close region sweep, described in the next section) will look at a large number of potential fragments to determine the next block to build the file and the falsely eliminated block will be reconsidered.

6. Refined file carving

As mentioned earlier, Garfinkel shows that files with greater than four fragments are rare, and file systems will almost never fragment a file on a block by block basis. What this means is that fragments typically consist of tens if not hundreds or thousands of blocks. We, therefore, propose a carving system that takes into account the fact that fragments typically consist of multiple blocks. In essence we believe that a file carver needs to find the fragmentation points of each file by sequentially testing each block after the header block. So our refined file carver will then contain the following steps:

(1) Identify starting block of file.
(2) Sequentially check each block after first and determine fragmentation point/file end.
(3) If fragmentation point is detected, find starting point of next fragment.
(4) Continute with step 2 from starting point of next fragment.

The first step in the new file carving system is the same as the traditional step for file carving – identify the data blocks that include the file header. The information from this initial data block is then utilized to recover the base-fragment through analysis of the neighboring blocks until a fragmentation point is detected. Once the fragmentation point is determined the starting block of the subsequent fragment must be determined. Using information from the file recovered so far the next fragmentation point (if the file has more than two fragments) has to be identified. This process is repeated until the file is completely or partially recovered from all the available fragments. Based on this we have modified the PUP algorithm as described earlier with SHT.

6.1. SHT–PUP

Recall that PUP was an algorithm developed to build multiple files in parallel. Its major drawback was that it did not take into account the fact, that fragments typically consist of many blocks. We have developed a solution for file carving fragmented files using SHT and PUP, called SHT–PUP. SHT–PUP begins by choosing all available headers of a file-type and attempting to use sequential hypothesis testing as described above to determine the base-fragment and fragmentation point. Once the fragmentation point of each base-fragment is identified, the remaining available blocks are classified into bins on the basis of file type. This is realized by the use of keyword and statistical tests.

Keyword tests utilize the Aho–Corasick algorithm (Aho and Corasick, 1975), which is a string searching algorithm that locates elements of a finite set of strings within an input text. It matches all patterns "at once", so the complexity of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches. The statistical tests are exactly the same described in Section 4 and involve entropy of the data and the OSCAR method (Karresand and Shahmehri, 2006a,b).

It is shown in Garfinkel (2007) that the gap between fragments is rarely more than 80 blocks and that the majority of the time the gap is much smaller. In fact a gap size of 8 blocks occurs in 4327 files and the next closest was a gap size of 32 blocks that occurs in 1519 blocks. As a result, we developed a process called the *close region sweep* to determine the starting point of the next fragment. For each base-fragment and

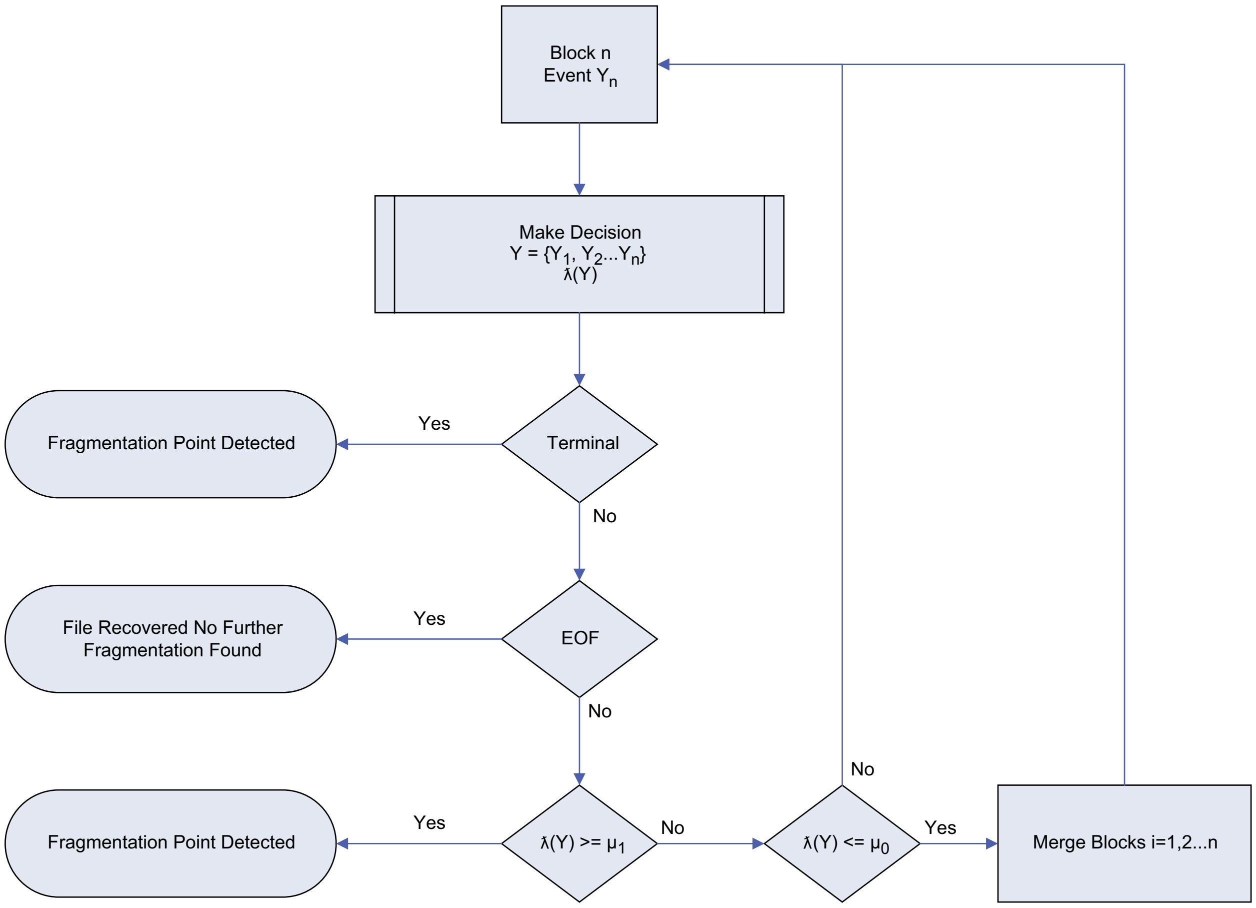

Fig. 6 – Flow diagram of fragmentation point detection algorithm.

fragmentation point identified, we look at the bin of the file-type and attempt to decode each of the blocks in an area of about 5000 blocks next to the base-fragment. For each file being recovered we store the top 20 block results based on our matching metric. If file *i* is the file with the best of the 20 results say block b_x, then we merge this block to the base-fragment of i. We then proceed with sequential hypothesis testing again until, another fragmentation point is found. We then store the best 20 results for the fragmentation point exactly as we described earlier. We repeat this process until all files are built or have failed in finding good candidates.

SHT–PUP has the advantage of only serially comparing blocks until a fragmentation point is detected, and attempting to then continue from the "best" of available fragmentation points of all available files. Assuming a good matching metric, files with blocks yielding better matching metric scores, will be built first, thus reducing the blocks required to analyze for the other file recoveries.

6.2. *Bifragment gap carving with SHT*

In addition, fragmentation point detection can also be used to enhance the efficiency of BGC. Recall that starting with a gap size of 1 all possibilities of blocks between a header and footer are carved. If b_f is the ending point of the first fragment and b_s is the starting block of the next fragment, every combination of b_f starting with b_f greater than b_{h+1} (block after header), needs to be evaluated. By utilizing fragmentation point detection, we can improve this by starting b_f at b_a where b_a is the detected fragmentation point. For base-fragments with hundreds of files this dramatically improves the performance of this algorithm.

7. Experiments and results

In this section, we demonstrate the use of the proposed fragmentation point detection algorithm by focusing on JPEG file recovery. Essentially this requires designating a proper matching metric, i.e., Y, and obtaining the conditional probabilities under two hypotheses, i.e., $Pr(Y|H_1)$ and $Pr(Y|H_0)$, as defined in Section 4. We will then present the performance results of the method for DFRWS 2006 and DFRWS 2007 test-sets.

7.1. *Choice of matching metric*

A matching metric is required to determine the likelihood that a block follows another in the correct reconstruction of a file. The metric is used to calculate the conditional probabilities in the above mentioned two hypotheses that indicate a block follows another. Motivated by the fact that local smoothness

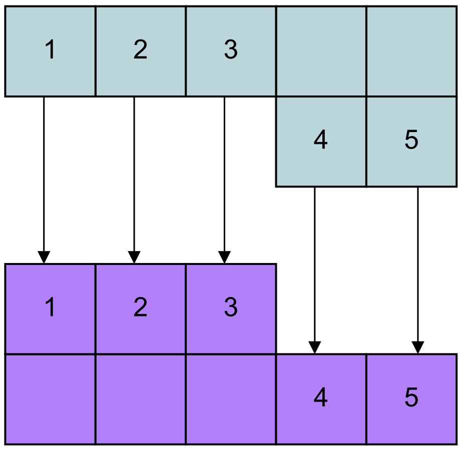

Fig. 7 – Simple example of which pixels are compared against which pixels.

assumptions often hold for natural images, Memon et al. (Pal and Memon, 2006; Pal et al., 2003; Shanmugasundaram and Memon, 2003) propose to use the discontinuity between two neighboring image blocks as a means to determine the likelihood of a block following another. This is realized by computing the sum of differences across the RGB pixel values between the edges of two image blocks and is then normalized by the number of pixels used in the calculation. This returns a score between 0 and 1 where a score is assumed to indicate a better match. Fig. 7 shows a very simple example of an image only five pixels long and two blocks being compared for this image.

7.2. *Estimation of model probabilities*

The conditional probability density functions for the matching-metric Y, $P(Y|H_0)$ and $P(Y|H_1)$, are obtained empirically from two training sets of 600 images that were randomly collected and had varying JPEG compression ratios and resolutions. (It should be noted that the training sets did not include images included in DFRWS 2006 or 2007 test-sets.) In obtaining $P(Y|H_0)$, model for fragments that belong together, we measured the variation in Y by computing it for each block based on the matching metric in each image with its correct next block. For the model where fragments are merged incorrectly, we randomly took three or more fragments from each image and attempted to find 10 other blocks from other images that decoded without an error and obtained observations of Y based on the matching metric only for those blocks that can be decoded correctly. Fig. 8 provides the conditional probabilities. It can be seen that the resulting Y values in both cases are significantly apart from each other. We set the thresholds desiring a detection rate of 0.7 and false elimination rate of 10^{-15}.

While we use existing models to determine $Pr(Y|H_0)$, we believe that a further improvement can be achieved by dynamically building a model for each image. By evaluating pixel-wise differences in the recovered parts of an image, we can build and refine the model by also updating it with the subsequently merged blocks.

7.3. *Results*

Using these models, the proposed method on fragmentation point detection, as described in Fig. 6, is applied to recover JPEG images in DFRWS 2006 and 2007 test cases. We must note again that during fragmentation point detection false-eliminations (i.e., a fragmentation point is identified before the actual fragmentation point) are preferable over false-additions (i.e., fragmentation point identified belongs to another file). This is because the second step of the carving in PUP and BGC can look at a large number of potential fragments to determine the next block to build the file and the falsely eliminated blocks will be in contention for being chosen. On the other hand, manual intervention is required to recover from a false addition.

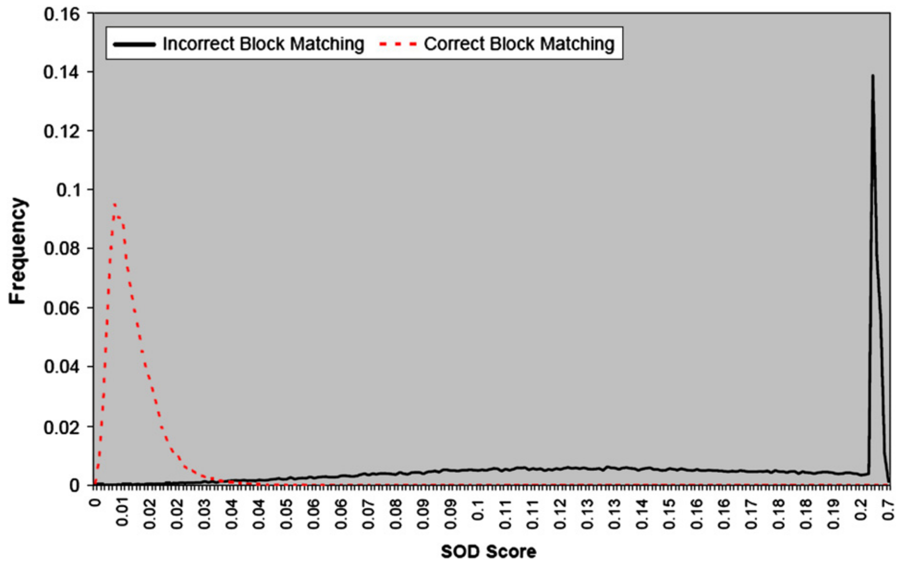

Fig. 8 – Model for both correct and incorrect mergings of JPEG blocks.

Both the DFRWS 2006 and 2007 challenge test-sets lack a file table, therefore, the default block size is set to 512 bytes. The DFRWS 2006 challenge test set consists of 14 JPEG images of which only 7 are fragmented. Each of the seven fragmented JPEGs have only two fragments (i.e., are bi-fragmented). In addition, the second fragment of each fragmented image is in front of the starting fragment (i.e., had a greater block number than the starting fragment). The DFRWS 2007 challenge test set contains 19 JPEGs only one of which is not fragmented. In addition, a few of the JPEGs have three or more fragments. One of the JPEGs has six fragments. To make the reassembly problem even harder, some fragments are stored in a location behind the correct fragment that precedes it. In other words, the fragmentation point of a fragment has a block number greater than the starting point of the next fragment. This makes recovery much trickier and as a result our close region sweep aspect of the SHT–PUP algorithm looks after and *before* the fragmentation point. Finally, some of the JPEGs have deliberately introduced errors causing our decoder to fail. Some JPEGs also have missing fragments. We do not attempt to correct or recover from decoding errors or missing fragments.

We now present the results for both SHT–PUP, BGC and BGC with SHT, SHT–BGC. It should be noted that we initially implemented BGC based solely on information provided in Garfinkel (2007). However, the resulting implementation had a running time of 80 min for recovering all images in DFRWS 2006. We then modified, the algorithm to detect the first block that failed to decode after the header of an image, and ensured that b_f (the ending point of the base-fragment) was checked starting from the block prior to the one that failed to decode. Then for each legitimate gap size, we moved b_f to the left until it reached the header. This reduces the running time to 22 min for DFRWS 2006 test-set. This, technique when utilized with SHT, reduces the running time to 5 s. An alternative implementation for BGC is instead to start b_f at the block before the decoding error as described earlier and try all possible gap sizes before moving left for the new b_f.

Even without SHT this implementation takes only 11 s to recover all images from DFWRS 2006 test-set as compared to SHT–BGC which took 5 s. Results are obtained based on the latter described implementation of BGC.

Table 2 provides the results of recovery using *Decoding Detection*, *BGC* as well as *SHT–PUP*. Decoding detection is nothing more than decoding until an error occurs or the image is built. While this cannot recover fragmented images, it is important to show how it also fails to identify fragmentation points correctly and even more importantly, how it causes false additions (8 in 2007 and 1 in 2006). Finally, decoding detection does not cause false eliminations since it merges all blocks that can be successfully decoded.

In contrast, our SHT–PUP causes zero false additions, however, there are a total of five false eliminations. All five of the falsely eliminated images are fully recovered, and this is because the close region sweep phase correctly identifies that the first falsely eliminated block for each fragment belongs to the fragment (it has the best score of the other 5000 blocks checked). It should be noted that due to deliberately introduced decoding errors and missing blocks some JPEGs are unable to be fully recovered, however, in all but one case we are able to achieve the best possible construction.

Table 2 – Recovery results for DFRWS 2006 and 2007 test-sets

DFRWS	2006	2007
Total images	14	18
Fragmented images	7	17
Un-fragmented images	7	1
Decoding detection		
Recovered un-fragmented	7	1
False elimination	0	0
False addition	1	8
Fragmentation point detected	5	9
Recovered fragmented	0	0
Total recovered	7	1
Bi-fragmented gap carving		
Recovered un-fragmented	7	1
False elimination	0	0
False addition	1	8
Fragmentation point detected	5	9
Recovered fragmented	0	2
Total recovered	14	1
Time taken	9 s	Hours
Time with SHT	5 s	Hours
SHT PUP		
Recovered un-fragmented	7	1
False elimination	2	3
False addition	0	0
Fragmentation point detected	5	14
Recovered fragmented	7	16
Total recovered	14	17
Time taken	13 s	3.6 min

It is also interesting to note that three of the falsely eliminated images have a detected fragmentation point exactly one block prior to the actual fragmentation point (false elimination). A closer inspection reveals that these cases occur when the test for a block is inconclusive and the next block causes a decoding error. Our method by default discards blocks that are undecided if a validation fails to avoid a false-addition. These blocks are later detected to belong to correct fragments during the recovery phase. In DFRWS 2006, we are able to recover all seven fragmented images without a problem. In DFRWS 2007 we are able to recover all but two decodable (without errors) fragmented images without manual intervention. One of the two images, is also manually recoverable by identifying the incorrect block and simply stating that it should not considered for the next reassembly iteration.

8. Conclusion

In this paper, we have demonstrated the effectiveness of using sequential hypothesis testing for identifying the fragmentation point for JPEGs. We have shown the advantages of true content based recovery compared to techniques that solely utilize decoding or file structural errors for recovery. Additional work will be done in building and refining the models for a file as it is being built. In future work, we plan to create models for other file formats, like those of Microsoft Office, Email (PPT files), etc. In addition, we wish to improve the accuracy and efficiency of the close region sweep phase by identifying the most likely candidates for the next fragment's starting point. Finally additional work needs to be conducted in recovering from decoding errors as well as identifying and handling missing fragments.

REFERENCES

Aho, Corasick M. Fast pattern matching: an aid to bibliographic search. Communications on ACM June 1975;18(6):333–40.

Amiga Smart Filesystem. http://www.xs4all.nl/hjohn/SFS.

Carrier Brian. File system forensic analysis. Pearson Education; March 2005.

Carrier B, Wietse V, Eoghan C. File carving challenge 2006, http://www.dfrws.org/2006/challenge; 2006.

Carrier B, Wietse V, Eoghan C. File carving challenge 2007, http://www.dfrws.org/2007/challenge; 2007.

Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik 1959;1:269–71.

Garfinkel S. Carving contiguous and fragmented files with fast object validation. In: Proceedings of the 2007 digital forensics research workshop, DFRWS, Pittsburgh, PA; August 2007.

Karresand Martin, Shahmehri Nahid. Oscar – file type identification of binary data in disk clusters and RAM pages. IFIP Security and Privacy in Dynamic Environments 2006;201: 413–24.

Karresand Martin, Shahmehri Nahid. File type identification of data fragments by their binary structure. IEEE Information Assurance Workshop June 2006:140–7.

McVoy LW, Kleiman SR. Extent-like performance from a UNIX file system. In: Proceedings of USENIX winter'91, Dallas, TX; 1991, p. 33–43.

Pal A, Memon N. Automated reassembly of file fragmented images using greedy algorithms. IEEE Transactions on Image processing February 2006:385–93.

Pal A, Shanmugasundaram K, Memon N. Reassembing image fragments. In: Proceedings ICASSP, Hong Kong; April 2003.

Richard III Golden G, Roussev V. Scalpel: a frugal, high performance file carver. In: Proceedings of the 2005 digital forensics research workshop, DFRWS; August 2005.

Shanmugasundaram K, Memon N. Automatic reassembly of document fragments via context based statistical models. In: Annual computer security applications conference, Las Vegas, Nevada; 2003.

STORAGEsearch.com. Data recovery from flash SSDs? http://www.storagesearch.com/recovery.html.

Sweeney A, Doucette D, Hu W, Anderson C, Nishimoto M, Peck G. Scalability in the XFS file system. In: Proceedings of the USENIX 1996 annual technical conference, San Diego, CA; 1996.

Wald A. Sequential analysis. New York: Dover; 1947.

Anandabrata Pal is doing his PhD degree in Computer Science at Polytechnic University. He is specializing in File Carving and Data Recovery. His research interests also include code obfuscation and file system forensics.

Husrev T. Sencar received his PhD degree in electrical engineering from New Jersey Institute of Technology in 2004. He is currently a postdoctoral researcher with Information Systems and Internet Security Laboratory of Polytechnic University, Brooklyn, NY. His research interests lie in security of multimedia and communication.

Nasir Memon is a Professor in the Computer Science Department at Polytechnic University, New York. He is the director of the Information Systems and Internet Security (ISIS) lab at Polytechnic. His research interests include Data Compression, Computer and Network Security, Digital Forensics, and Multimedia Data Security.