Image Attribute Estimation for Forensic Image Reconstruction from Fragments

Kevin Montambault
University of Massachusetts Dartmouth
kmontambault@umassd.edu

Gökhan Kul
University of Massachusetts Dartmouth
gkul@umassd.edu

Abstract

The increasing prevalence of cyber-crime has led to a surge of new forensics tools aimed at collecting digital evidence from a suspect's computer. A suspect's hard drive can be the largest source of collected information, but the task of collection can be made significantly more difficult when the contents of a hard drive are deleted or damaged. In these circumstances the information needed to read files normally may be missing, leaving only the raw, often fragmented, data behind. If we were able to reliably reconstruct files from this raw data, then it would be more difficult for suspects to destroy potential evidence. In this paper, we focus on the reconstruction of an image from a set of fragments. This research contributes a novel image reconstruction method which utilizes pre-stitch data extraction on individual data sectors. We show that, when certain attributes are successfully extracted from the data sectors, this method yields a high reconstruction accuracy even when used with a naïve stitching algorithm on heavily fragmented image files.

Keywords: image reconstruction, image forensics, image attribute estimation, stitching

1. Introduction

As cyber-crime continue to become more pervasive, it remains critical that forensics investigators can reliably collect digital evidence for use in these cases. The term digital evidence can refer to evidence collected from many different sources, but one particularly large source of digital evidence is a suspect's hard drive (Casey (2011)). In some circumstances, however, it is possible that the suspect has deleted, corrupted, or otherwise damaged a hard drive in such a way to make accessing the data normally impossible. Although the data may no longer be accessible through a file system, it may still be possible to extract files from these drives using a technique called file carving. File carving is the process of scanning a hard drive's raw data with the goal of reconstructing the contents of individual files from the disk without relying on the underlying file system (Poisel and Tjoa (2013)). This process is made significantly more difficult when the disk's data is fragmented, meaning a single file is saved in sections at non-continuous locations across the disk. Reconstructing these files requires the use of special algorithms which will henceforth be referred to as stitching algorithms (Casey (2011)).

Because end-to-end file carving can be a lengthy process, this paper will focus solely on the final stages of file carving. We assume that through previous methods, sectors have been clustered together based on which images they belong to, and that RGB information is available for each sector. These are safe assumptions to make, as research has already been done surrounding the clustering and classification of file fragments (Ali and Mohamad (2021) and Tsamoura and Pitas (2009)), and RGB information is readily available for image types like BMP (Microsoft (2021b)), some raw formats, and even JPEG (Sencar and Memon (2009)). We also assume that the bit depth of the target images will be divisible by 8, allowing us to consider only a whole number of bytes as potential pixels. This is to reduce the complexity of the implementation of the proposed algorithm by allowing us to avoid manipulating individual bits for comparisons. This change only reduces the complexity of a single stage in the pipeline and has little impact on the function of the algorithm as a whole. Our testing data will consist of sets of completely scrambled image fragments stored as raw RGB data. Reconstructing raw RGB data allows the resulting algorithm to be generalized to any file format where RGB data can be extracted, as mentioned previously. Raw RGB data is also extremely similar to 24-bit color BMP files, which were used by Pal et al. (2003) to test their reconstruction algorithm. The only difference between the two formats is the lack of a 54-byte header at the beginning of each file, and lack of 0-3 bytes of padding at the end of each scanline (Memon and Pal (2006) and Microsoft (2021b)).

A major difference between our proposed algorithm

URI: https://hdl.handle.net/10125/103436
978-0-9981331-6-4
(CC BY-NC-ND 4.0)

and existing solutions is the fact that our algorithm considers each sector as an individual fragment regardless of its neighbors. This differs from existing algorithms which operate around large fragments of image data stored across many continuous sectors (Memon and Pal (2006), Pal et al. (2003), and Tang et al. (2016)). By focusing our algorithm on reconstructing from individual sectors, we can create a robust method against small image fragments, missing or damaged sectors, and heavily fragmented images.

2. Background

In this section, we describe the process of file fragmentation and specific vocabulary like byte depth and sectors.

Byte Depth. The more common term *bit depth* is the number of bits per pixel a specific file uses to store an image (Microsoft (2021a)). For simplicity, this paper focuses only on those images whose bit depth is divisible by 8, so, we will be referring to this attribute as *byte depth*. Because there are exactly 8 bits in a byte, an image with a bit depth of 24 can be said to have a byte depth of 3. The term *byte depth* will be used extensively throughout this paper and will always refer to the bit depth of an image divided by 8.

Sector Size. The sector size of a disk is simply the smallest amount of data that can be read or written to the disk at a single time. This amount is measured in bytes and is typically a power of two ranging from 512 to 4096 (Garfinkel (2007)).

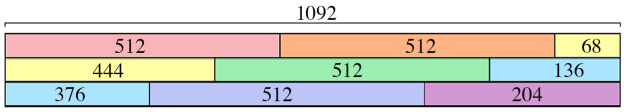


Figure 1: How seven continuous 512-byte sectors are positioned relative to each other in an image with a width of 1092 bytes

Width Offset. Width offset is a non-standard term which is used throughout this paper for lack of a more suitable alternative. It refers to the *staggered* nature of sectors when viewing their positions in a 2-dimensional image. This staggering results from a mismatch between the width of the image and the sector size, which can be seen in Fig. 1. More specifically, it refers to the number of bytes that vertically-neighboring sectors are offset by relative to one another. If the width of the image, measured in bytes, is not evenly divisible by the sector size, then subsequent lines will be offset by the remainder. In Fig. 1, the width offset can be said to be either +444, or -68, but to reduce complexity, offsets will be limited to half of the sector size in the positive and negative direction. In the above example, this leaves the width offset to be -68 bytes.

File Fragmentation. A file becomes fragmented when it is saved in pieces to non-continuous locations on a hard drive (Sari and Mohamad (2020)). This process can be seen in Fig. 1, where file sectors are being saved to the drive at the first available locations. Fragmentation can also occur if data is appended to the end of a file, or if the operating system itself places constraints on how files can be saved (Garfinkel (2007)). The former can be seen in Fig. 2, and the latter can be seen in Fig. 3.

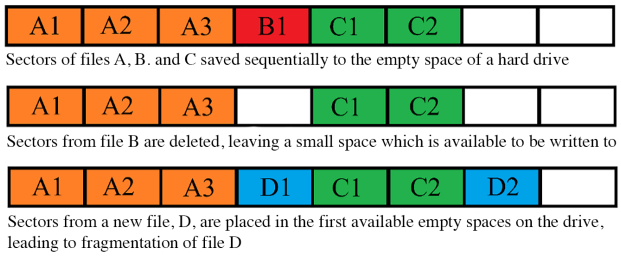


Figure 2: The process of file fragmentation as a result of deleting and creating files of varying sizes

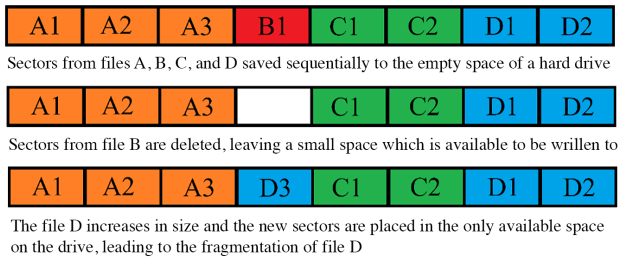


Figure 3: The process of file fragmentation as a result of an existing file growing in size to fill an existing space

Natural Image. A natural image is any image which has high spatial covariance (Lythgoe and Partridge (1989)). For example, pictures taken of a real-life scene using a digital camera will yield an image where neighboring pixels are highly similar, making them a better candidate for content-based image reconstruction. This is in contrast to artificial images which are generated or modified on a computer and can have abrupt content changes.

3. Related Work

The problem of image reconstruction is typically approached as an ordering problem which seeks to minimize the distance between the boundaries of sequential sectors (Memon and Pal (2006), Pal et al. (2003), Pal et al. (2008), and Tang et al. (2016)). In these approaches, it is often the distance metric or specific ordering algorithm which is modified. Methods that utilize this technique are susceptible to error when input images have sudden changes in content, such as the sharp edge of a building against a clear sky, as mentioned by Tang et al. (2016). These methods

perform poorly under these conditions due to their heavy reliance on a small number of pixels directly adjacent to sector boundaries. This work hopes to avoid this problem by comparing sectors vertically, thus allowing for sectors to be compared along their entire length. This comes with a host of other challenges, which will be discussed in detail within the methodology section. Pal et al. (2003) utilize a graph-based approach for the ordering algorithm and compare their results with a greedy alternative, similar to the ordering method used in this paper. They concluded that greedy approaches perform well despite their simplicity, but still provide a lower reconstruction accuracy verses a more thorough graph-based approach. Another graph-based approach is used by Memon and Pal (2006), however they are attempting to stitch large fragments directly on top of one another instead of horizontally. The same stitching technique is used by Tang et al. (2016), who proposed a new metric for evaluating potential fragment stitches. Both of these methods rely on large continuous fragments as well as prior knowledge of the image width. The width information was obtained from header fragments, but this technique limits the stitching algorithm to only growing from header fragments because of their known widths. There are several methods that aim to classify file fragments based on file types (Chen et al. (2018)). These methods attempt to extract high-level features from the file fragments, which can be useful for deciding how similar a pair of fragments are. While these methods focus on differentiating file types, they could be adapted cluster image fragments based on the images that they originated from. Chen et al. (2018) even convert various file fragments into images before classifying them. Tang et al. (2016) demonstrate a new algorithm for scoring potential fragment stitches together. They show how their method greatly out-performs commonly used methods like Euclidian Distance and Sum of Differences. Their algorithm utilizes a second scanline of pixels from the image, which requires extra knowledge about the image such as the width. This method would also not be suited for particularly small fragments.

Karresand et al. (2019) work to improve the efficiency of existing file carving algorithms by showing that portions of a disk's NTSF partitions can be prioritized by creating a probability map of finding unique data across the partition. The methods mentioned here are aimed at either clustering fragments based on content or improving efficiency of existing file carving methods. Accurate clustering methods would provide a more robust input to the proposed pipeline, and more efficient reconstruction methods would decrease the time required to stitch sectors in one of the pipeline stages. These methods alone, however, still share the same shortcomings of the traditional carving algorithms that they are improving.

4. Methodology

The goal of this research is to develop an algorithm which accurately reconstructs an image from a set of scrambled fragments by utilizing meaningful image attributes extracted prior to the final reconstruction. The proposed algorithm will be split into four major stages. The purpose and implementation of each stage will be described in Section 4.1 and 4.2 respectively.

4.1. Algorithm Pipeline

This section will detail the four major stages of the proposed pipeline. For each stage, we discuss what information is gained and how that information can be used either towards future stages or for the final reconstruction. This can be seen in Fig. 4, which outlines each of the four major stages as well as the information gained after each step. As we can see, the proposed algorithm will alternate between attribute extraction and sector stitching until the final reconstruction is complete. As input, the algorithm will take in a set of sectors collected from an image and scrambled. The reconstructed output will be in the form of a single image file with the BMP file format.

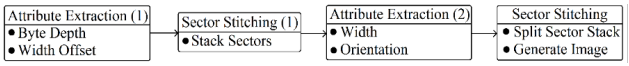


Figure 4: The four stages of the proposed algorithm pipeline, including what is gained at each stage

Stage 1: Attribute Extraction (1). The goal of the first attribute extraction phase is to determine the byte depth and width offset of the source image (see Fig. 4). These two attributes are crucial for discovering the values of future attributes, most importantly the true width of the source. The byte depth can be used to estimate the width by narrowing down the number of possible widths of the image. This can be done by only considering those widths which divide evenly by the byte depth, since a scanline of an image must end at the boundary between two pixels. Similarly to the byte depth, the width offset can also be used to reduce the number of possible widths. By definition, the width offset is the sector size minus the number of bytes from a single sector which overflow onto the next scanline. This means that the width (in bytes) of the image must be the width offset plus some whole number times the sector size. This is detailed further in the Stage 3 of the implementations section but means that the total number of possible widths can be reduced even further to only those widths that match the stated equality. The width offset has an

additional purpose, in that it allows us to only compare sectors at that specific offset when performing sector ordering. Since all sectors share the same offset relative to their neighbors, as shown in Fig. 1, there is no need to compare sectors at any other offset. This dramatically reduces the amount of comparisons needed between sectors by a factor of the sector size. This fact will be relevant to the implementation of Stage 2 of the pipeline.

Stage 2: Sector Stitching (1). Stage 2 of the pipeline focuses on vertically ordering, or *stacking*, the input set of scrambled sectors. This is the main reconstruction step and determines the order of sectors vertically in the output image. The goal is to order the sectors in such a way that the sum distance between adjacent sectors is minimized, which is the same problem as The Traveling Salesman problem (Black et al. (1998)). This ensures that the content of each sector is as similar as possible to its closest neighbors, which is, as previously described, an attribute of natural images. Shown in Fig. 5 is one possible stacking order of the sectors present in Fig.1. We call this ordering *perfect* because each sector has the correct neighbor. There are multiple perfect ways to order the sectors vertically, which will be discussed in more detail in the Stage 3 section.

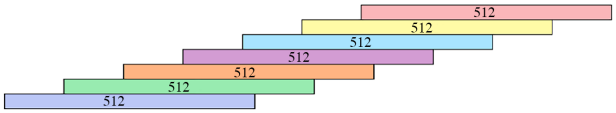


Figure 5: A series of sectors stacked on top of one another. The stack shown is one of two possible perfect stackings of the sectors shown in Fig. 1

Stage 3: Attribute Extraction (2). The goal of this step is to determine both the width of the true image as well as the orientation of the sector stack generated in Stage 2. The incoming sector stack will also be split into multiple smaller stacks, which represent vertical strips of the image (see Fig. 6). The width of the image is a critical piece of information that is required for generating the final image file. Additionally, knowing the width of the image also allows us to roughly calculate the height based on how much data is present.

Splitting the sector stack incoming from Stage 2 is necessary for creating the final reconstructed image because the smaller stacks must be placed next to one another to create the final 2-dimensional image. This placement is done in Stage 4 and is the final step of reconstruction. Each smaller stack represents a vertical slice of the image, where no slice contains sectors that overlap more than 50% of any sector from another slice. Shown in Fig. 6 are the vertical slices of the image showed in Fig. 1. These two smaller stacks were obtained by splitting the single stack shown in Fig. 5 between the purple and orange sectors. There is another orientation that Stage 2 could have produced, and that would be the orange, green, and blue sectors stacked on top of the pink, yellow, blue, and purple sectors. This would be another perfect stacking of the example sectors, as an alternative to the stack shown in Fig. 5. Stage 3 should produce the same output regardless of which variation is generated by Stage 2.

Figure 6: How the sector stack can be split into two separate sector stacks, each representing a vertical strip of the original image

Stage 4: Sector Stitching (2). The final stitching step will involve ordering the individual sector stacks horizontally to create a 2-dimensional image. The output of this process can be seen in Fig. 7. It is then a simple matter of writing the sectors to the image file from left to right, top to bottom. Even though the output in Fig. 7 does not appear square, written as raw data to an image file will produce a square image.

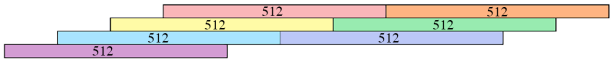


Figure 7: Sector stacks stitched horizontally into a 2D image

4.2. Implementation

In this section, we cover the specific implementation details for each stage in the reconstruction pipeline. Each stage is broken up into parts based on the discrete operations included in this stage (see Fig. 4). Included for each implementation is the goal, specific function to be optimized in the proposed solution, and possible alternatives.

Stage 1 - Attribute Extraction (1). *Byte Depth* The byte depth of an image can be determined very confidently by analyzing the frequencies of neighboring bytes. As described by Lythgoe and Partridge (1989), natural images contain high spatial covariance, meaning neighboring pixels will typically have similar colors. This fact can be leveraged to discover how many bytes are used to represent each pixel by testing various byte depths to discover the one which yields the highest covariance between neighboring pixels. Other methods that could discover the byte depth would be a Fourier transform, or a similar auto-correlation operation. These methods need only test a small set of common byte depths like 2, 3, 4, and 6 (Microsoft (2021a)). *Width Offset:* Our implementation of the width offset estimator utilizes a two-step process. The first step determines

a small set of sector pairs, where sectors in a pair are close to each other. Then, the optimal width offset for each pair is found and cataloged. The width offset with the largest number of occurrences is selected as the width offset. As mentioned, the first step is to obtain a set of sector pairs, where the sectors in each pair are spatially close to each other. This was necessary in our implementation to reduce the number of comparisons needed to brute-force an optimal width offset during the next step. We can dramatically reduce the number of comparisons by first utilizing this rough distancing step. There are many options to choose from for this step, but our implementation utilizes a Fourier transform in order to compare the similarity of component frequencies for each sector. This has the advantage of being largely position-independent (the horizontal position of a feature can be discarded) while still maintaining high dimensionality. Many other vectorization functions could replace the Fourier transform in this step. Simple examples would be variance, entropy, or average color. Alternatively, the outputs of clustering models similar to Chen et al. (2018) and McDaniel and Heydari (2003) could be re-used here for free if that algorithm was already used to obtain the list of image sectors to begin with. For each pair in this set, the offset which minimizes the Euclidian distance between the two sectors is logged. Once this is completed for all selected pairs, the offset that obtained the highest frequency is selected as the width offset. This process is formally defined by maximizing the below function F for o, where S is the sector size, o is a given offset, and n is the number of pairs selected. P is a pair of sectors who have been determined to be spatially close by one another based on the rough distance function discussed in the previous step.

$$G(A, B, o) = \sum_{i}^{S-o} (A_{i+o} - B_i)^2$$

$$F(o) = \sum_{i}^{n} MIN(G(P_{n,1}, P_{n,2}, o), G(P_{n,2}, P_{n,1}, 0))$$

To reduce complexity, the absolute value of the width offset is used through the remainder of the algorithm. Ignoring the sign of the offset has the effect of flipping the image vertically. Ignoring the sign of the offset benefits this step because it allows us to ignore which sector in the pair is on top, since we are just interested in the relationship between the two. This would be a problem for the final reconstruction, but there is no guarantee that the image will be reconstructed in Stage 2 in the correct orientation to begin with. This means that the orientation correction in Stage 3 will need to be done regardless, so there is no harm in discarding the sign of the offset in this step.

Stage 2 - Sector Stitching (1). *Sector Stacking* The goal of sector stacking is to order the sectors vertically in such a way that the sum distance between neighboring sectors is minimized. Our algorithm utilizes a greedy approach, and simply stitches the current closest pair of sectors until all have been stitched. This is the costliest portion of the algorithm, with a time complexity if O(n^3) relative to the sector count. We only require one comparison between each sector, since each sector is compared only at the width offset calculated in the previous step. The function G, mentioned in the previous section, is used as the comparison function in this step where o is set to the width offset previously determined by Stage 1. There are algorithms that can perform this task in shorter time, such as finding the pair of closest points in O($n \cdot logn$) time as demonstrated by Shamos and Hoey (1975), but the efficiency of the implemented algorithm, while considered, was not the primary focus of this work.

Stage 3 - Attribute Extraction (2). *Width Estimation* The goal of width estimation is to determine the true width of the original image. This can be done using the byte depth, width offset, and the stack of sectors obtained in the previous stage. Because a scanline must end on the boundary between two pixels, we can greatly reduce the number of possible widths to only those where some multiple of the sector size plus the width offset are a whole multiple of the byte depth. The minimum width of the image can be discovered by satisfying the equality below where S is the sector size, o is the width offset calculated previously, d is the byte depth, and n is some positive integer.

$$0 = mod(nS + o, d)$$

After determining the smallest value for n which satisfies the equality, the minimum width is defined by the equation bellow. Because we don't know whether the stack is upside- down or not, two possible minimum widths must be calculated using both the positive and negative values of the width offset, o. As previously mentioned, using a negative value for o has the effect of flipping the stack vertically during the final stitching step. If S is larger than the true width, a larger multiple of the true width will be discovered instead.

$$minimumWidth = nS + o$$

By discovering the minimum width, we also discover all possible widths of the image. The true width of the image must occur at a position where Sx is divided evenly by the d where x is some positive integer. This will yield a set of potential widths that can be evaluated in the next step. Once the minimum width and width interval are both determined, all possible

widths beginning at the minimum width and ending at the number of bytes in the data will be tested using the function H, shown below, where S is the sector size, n is the number of sectors, w is the width to be tested, and m is the squared difference between pixels along a vertical line in the sector stack at the x offset of i.

$$H(w) = \sum_{i}^{Sn-w} m_i m_{i+w}$$

The rationale behind this approach is that the sector stack will contain regular breaks where several neighboring sectors reach the end of a scanline, creating a distinct vertical line in the image content on sector stack. These vertical lines will occur at intervals matching the true width of the image. Examples of this can be seen in Fig. 8. By testing all possible widths, we can select the width which yields the largest sum difference at these intervals.

Figure 8: A vertical break in content where one scanline ends and another begins within the sector stack. Image is taken from a sector stack generated using the proposed algorithm

Image Orientation The sector stack created in Stage 2 has the potential to be upside down relative to the true orientation of the image. This portion of the algorithm is dedicated to determining whether the sector stack should be flipped vertically or not (i.e. if the negative version of the width offset should be used or not). This process corrects for using the absolute value of the width offset, as mentioned in the Stage 1 implementation section. When discovering the width in the previous step, two sets of width increments were tested: one for the positive width offset and one for the negative and we selected the width based on which width yielded the maximum value of function H. The width that was selected will be associated either with the positive or negative width offset which will tell us the orientation of the sector stack. Examples showing the stitching for an upside-down stack are shown later in the results section.

Stage 4 - Sector Stitching (2). The goal of this step is to split the single continuous stack of sectors generated by Stage 2 into multiple stacks. These stacks represent the multiple vertical strips of the image which will be combined to create the final two-dimensional image. The number of vertical strips necessary is relative to the width of the image as well as the sector size. The number of strips needed to fill the image is simply calculated by dividing the suspected width of the image by the sector size, yielding how many sectors are needed to fill the image horizontally. This is the same number n which is used in the equality shown previously in the width estimation implementation section. The height of the image can then be calculated roughly by dividing the total number of sectors by the number of sectors needed to fill the width. This calculation should almost always yield a decimal number slightly over the true height, since the last sector of the image is most likely not completely filled with image data. The only circumstance that this wouldn't be the case is if the width divided evenly by the sector size. For this reason, the calculated height is floored to the nearest integer. The sector stack is then split at intervals of approximately this height to generate the vertical strips needed to reconstruct the two-dimensional image. A buffer of one scanline up or down is added, and the stack is split between the pair of sectors with the largest Euclidian distance among the three possibilities. The reason for this is that a vertical strip may not be exactly the calculated height due to the wrap- around nature of the sectors at the edges of the image, so the best location within a scanline is selected. The Euclidian distance is maximized here because we are looking for the location where the bottom of a vertical slice of the image meets the top of a vertical slice. This will hopefully lead to a large difference in content resulting from the vast difference in location between the sectors. An example of such a horizontal break can be seen in Fig. 9.

Figure 9: A horizontal break in content where sectors from the bottom of a vertical slice of the image meet sectors from the top of another vertical slice. Image is taken from a sector stack generated using the proposed algorithm

5. Experimental Evaluation

This section will show how the proposed algorithm performed on the selected data set. The images from the dataset were run through the proposed pipeline with accuracy measurements being taken for byte depth, width offset, width, and orientation. Results will first be shown and discussed regarding the entire pipeline, then broken up to show how each individual stage performed, independent of previous stages.

The accuracy function A is defined below, where s is the number of sectors in the original image and i is the number of inversions required to transform the output reconstruction into the original image.

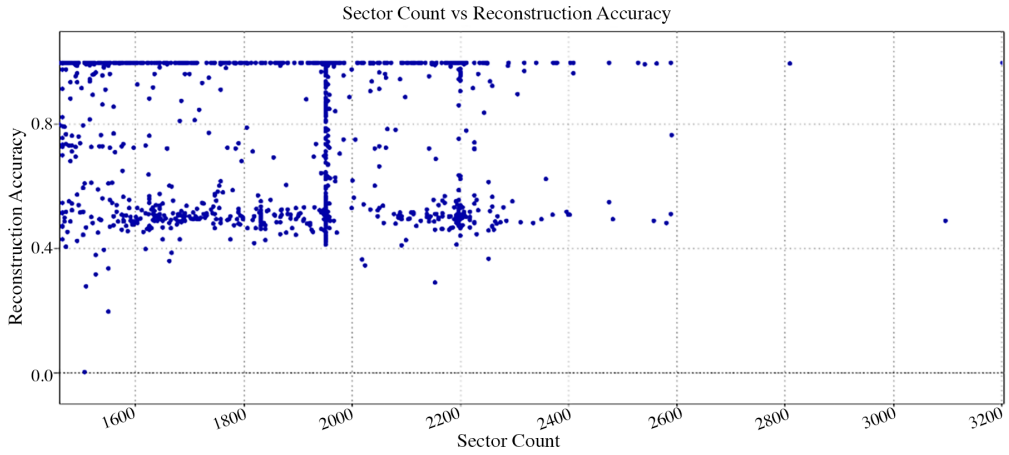


Figure 10: The relationship between the number of sectors in an image and the reconstruction accuracy

$$A(i,s) = 1 - \frac{2i}{s(s-1)}$$

5.1. Dataset

Due to the scope of this work, existing file carving datasets would not be suitable for testing the proposed algorithm. Many existing datasets have a focus on differing file types ("NIST, Forensic Images for File Carving" (2021)) and are made to test the capabilities of a fully implemented file carver. As previously mentioned, our work is intended for the later stages of carving and can be tested only on single images which require reconstruction. The dataset used to test the proposed algorithm consists of 1032 images of various types of flowers (Nilsback and Zisserman (2020)). All images included in this dataset are natural images taken with a camera. This is in contrast to images that may have been digitally created or manipulated using software. These images also often included large patches of low-noise data, such as clear sky, which provides an additional challenge for sector matching. The raw RGB values for each of these images was extracted then divided into sectors of 512 bytes each. This is the smallest sector size commonly found on hard drives (Garfinkel (2007)) and provides the greatest challenge for reconstruction. The sectors belonging to each individual image were scrambled independently of other images, resulting in 1332 individually scrambled images. Both the code[1] as well as the dataset[2] used are publicly available.

5.2. Complete Pipeline Results

Results for the complete pipeline can be seen in Column 1 of Table 1. We can see that the algorithm accuracy is 76.76% on average across all 1332 images in the data set. A total of 344 of the original 1332 tested images were able to be fully reconstructed with an accuracy of 100%, as can be seen in Column 5.

Table 1: The reconstruction accuracy for various portions of the pipeline

	Complete	**Stage 3-1**	**State 3-2**	**Stage 3-3**	**Ideal**
Images	1332	899	825	825	344
Sectors/Image	1870.48	1844.80	1850.56	1850.56	1830.38
Milliseconds/Sector	6.66	6.45	6.49	6.49	6.42
Byte Depth Accuracy	100.00%	100.00%	100.00%	100.00%	100.00%
Width Offset Accuracy	67.49%	100.00%	100.00%	100.00%	100.00%
Width Accuracy	61.94%	91.77%	100.00%	100.00%	100.00%
Orientation Accuracy	79.50%	95.44%	100.00%	100.00%	100.00%
Reconstruction Accuracy	76.76%	89.30%	92.89%	92.89%	100.00%

The decreasing average number of sectors in each column and the increasing reconstruction accuracy could encourage the assumption of a negative correlation between the two, but Fig. 10 shows how the number of sectors present in an image does not appear to largely impact the performance of the reconstruction algorithm. If there was a negative correlation between the two, as potentially indicated by the decreasing sector count as accuracy increases, we would expect to see a clear downward trend in Fig. 10.

5.3. Stage 1 - Attribute Extraction (1)

Stage 1 focused on determining the byte depth and width offset of the source image. The results for both of these attributes can be seen in the Column 1 of Table 1, but each will be discussed in detail within this section.

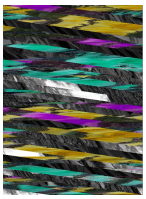
Byte Depth. We can see that the byte depth of all 1332 images was correctly determined, indicated by the accuracy of 100%. This high accuracy is to be expected, since there are a very limited number of possible byte depths, as described in section 4.2. It is important to note, however, that the dataset consisted of only images with a byte depth of three bytes as a result of the file format used in the dataset.

Width Offset. As seen in Column 1 of Table 1,

[1]Link removed for anonymized submission

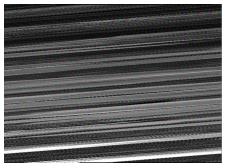
[2]https://www.robots.ox.ac.uk/ vgg/data/flowers/17

(a) Original Image



(c) A reconstructed image which used an incorrect width offset.

(b) An image which was reconstructed using the incorrect width



(d) An image which was reconstructed using the incorrect orientation

Figure 11: Original image and incorrect stitching results

the width offset was correctly determined in 67.49% of the 1332 tested images. We can see what happens to an image which is reconstructed using an incorrect image offset in Fig. 11.c. Note how the original image can somewhat be seen within the individual vertical strips (these are the sector stacks mentioned in Stage 4 of Section 4.2), but the strips are color shifted and miss-aligned. The color shift here is caused by sectors being offset incorrectly by a factor indivisible by the number of bytes used per pixel. This leaves some lines of the image to start in the middle of a pixel, leading to a miss-representation of the different color channels in the reconstructed image.

The major cause of incorrect offset estimations can be deduced from Figures 10, 11.c, and 12. Fig. 13 shows how the ideal offsets for the selected pairs of this particular image are distributed much more evenly across the possible offsets when compared to image used for Fig. 12. This could indicate that either the method used for selecting the ideal offset between two pairs is failing, or that the vectorization algorithm used to select close pairs is selecting sectors that are non-neighboring. We can confirm that it is the latter by looking at Fig. 14, which shows how the accuracy of the pairing algorithm significantly declines as the true width offset increases. Fig. 14 also shows that, when the pairing algorithm performs well, the ideal offset is able to be determined with little issue.

5.4. Stage 2 - Sector Stitching (1)

This stage of the pipeline attempted to stitch the sectors into a single continuous stack. Results for this portion of the pipeline were not isolated for lack of a meaningful performance metric. Stage 2 and Stage 4 both utilize the same greedy stitching algorithm

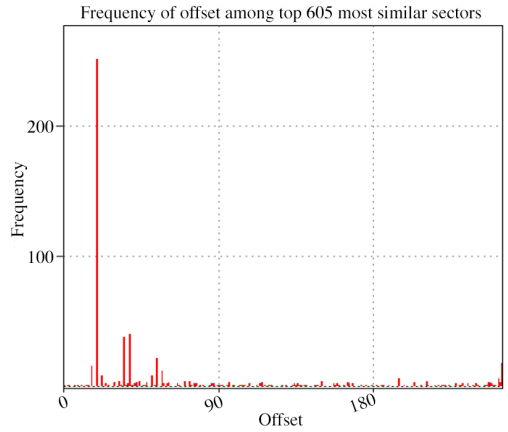


Figure 12: The frequency of ideal offsets between pairs of sectors in an image whose width offset was correctly determined. In this case, the correct offset was 19 which is shown as the large spike to the left of the graph. The number of pairs evaluated was equal to 30% of the total number of sectors, in this case 605 pairs

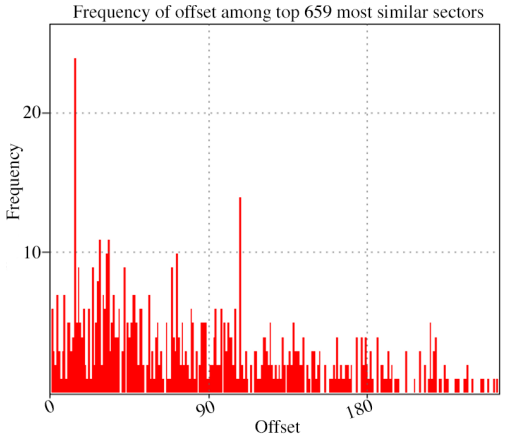


Figure 13: The frequency of ideal offsets between pairs of sectors in an image whose width offset was not correctly determined. In this case, the correct offset was 202. The number of pairs evaluated was equal to 30% of the total number of sectors, in this case 659 pairs

described in Stage 2 of Section 4.2, and both largely effect the final order of the sectors in the reconstructed image. For these reasons, they will both be evaluated when discussing the final reconstruction accuracy in the Stage 4 Results section.

5.5. Stage 3 - Attribute Extraction (2)

Stage 3 of the pipeline focused on extracting the width and orientation of the original image, described in Stage 3 of Section 4.1. The importance of correctly estimating the width and image orientation can be seen in the improved reconstruction accuracy between Column 2 and Column 3 of Table 1.

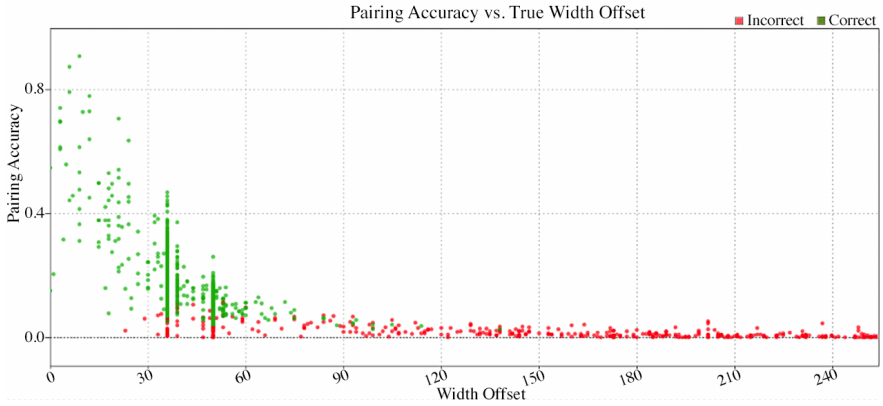


Figure 14: The relationships between the true width offset of a target image, the correctness of the extracted width offset, and the accuracy of the vectorization algorithm used to select close pairs. While it may appear that there are fewer **Correct** (green) points than **Incorrect** (red) points, this is simply because the Correct points are spaced much closer. Correct points make up 67% of the points shown on the figure, while Incorrect points make up the remaining 33%. Pairing accuracy, shown on the Y-axis, is the percent of pairs selected for the width offset estimation for a particular image were truly neighbors

Width. The goal of this step was to use the width offset estimation in combination with the byte depth and sector stack to estimate the true width of the image. Because this process uses the width offset, if the offset is determined incorrectly then the estimated width will also be incorrect. Of the 1332 reconstructions, there were no instances of the incorrect width offset yielding the correct width. When considering just those reconstructions which accurately predicted the width offset, the width was calculated correctly 91.77% of the time, as seen in Column 2 of Table 1. An example of how an incorrect width affects the resulting reconstruction can be seen in Fig. 11.b. In this case, the width used for reconstruction was smaller than the true image width. The image appears to be duplicated twice vertically, but these two halves are simply two different sets of neighboring sector stacks which are no longer interleaved with each other.

Orientation. As seen in Column 3 and Column 4 of Table 1, the accuracy of the orientation estimation is largely dependent on the accuracy of the width and width offset estimations. This is to be expected, since the orientation is determined using the suspected width of the image, as described Stage 3 of Section 4.2. The orientation is an important feature to determine correctly, because, unlike many of the other attributes, the reconstructed image will be completely unrecognizable if the incorrect orientation is used. An example of an image reconstructed using the incorrect orientation can be seen in Fig. 11.d.

5.6. Stage 4 - Attribute Extraction (2)

The goal of this stage was to reconstruct a 2-dimensional image using the width, orientation, and split sector stacks as described in Stage 4 of Section 4.2. As can be seen in Column 3 and 4 of Table 1, this stitching step was able to produce an average accuracy of 91.77% when all other attributes were correctly determined. This indicates that the greedy stitching approach works reasonably well.

6. Conclusion

The goal of this paper was to demonstrate the viability of a novel method for image reconstruction from completely scrambled fragments. We succeeded in showing that particular attributes such as image width, byte depth, and orientation can be determined using raw image data from the sectors, and that these attributes can be used to accurately reconstruct the source image. The utilization of this algorithm would allow image carvers to handle very small file fragments and heavily fragmented image data, as well as potentially allowing carvers to elegantly handle corrupted or missing sectors. This is possible because the proposed algorithm is able to treat individual sectors as independent from their neighbors. While direct comparisons cannot be made with existing tools due to the problem's scope, this work demonstrates that further research in this direction would be worthwhile. With some changes to the underlying algorithms, namely the sector vectorization method described in Stage 1 of Section 4.2 for reasons discussed in Section 5.2, the proposed pipeline could reach accuracies well over 90% as shown by the final column of Table 1.

Discussion. The methods proposed for extracting the various attributes are crude and un-optimized but provide a basis for future work. Each method for extracting specific attributes can potentially be

improved independently of other methods, meaning there are many opportunities for both efficiency and accuracy improvements. Similarly, the reconstruction algorithm itself is largely unoptimized. Our greedy implementation performs faster than a graph-based approach, but this potentially affects the accuracy of our reconstructions. That said, our results using a greedy algorithm performed reasonably well, and there is no guarantee the significantly increased runtime for a graph-based approach would yield significantly better reconstructions (Pal et al. (2003)).

Future work. A major benefit of this algorithm which was not pursued in this work is the ability to reconstruct images with corrupted or missing fragments. This is because the algorithm does not rely on the horizontal similarity between two neighboring sectors. If sectors are being stitched together horizontally, the gap between two sectors with a shared missing neighbor would be equal to the sector size. In a best-case scenario this would be a gap of 512 bytes. However, stitching sectors vertically, a missing sector would result in only a single-pixel gap between sectors with a shared missing neighbor. This significantly boosts the algorithm's ability to match sectors and can provide the ability to closely compare sectors that have many missing neighbors. Testing was limited to only images who were able to be fully reconstructed, but it would be unrealistic to assume this would always be the case when sectors are probabilistically clustered. We have already discussed how our current implementation could handle missing sectors.

References

Ali, R. R., & Mohamad, K. M. (2021). Rx_mykarve carving framework for reassembling complex fragmentations of jpeg images. *Journal of King Saud University-Computer and Information Sciences*, *33*(1), 21–32.

Black, P. E., et al. (1998). Traveling salesman - dictionary of algorithms and data structures.

Casey, E. (2011). *Digital evidence and computer crime: Forensic science, computers, and the internet*. Academic press.

Chen, Q., Liao, Q., Jiang, Z. L., Fang, J., Yiu, S., Xi, G., Li, R., Yi, Z., Wang, X., Hui, L. C., et al. (2018). File fragment classification using grayscale image conversion and deep learning in digital forensics. *2018 IEEE Security and Privacy Workshops (SPW)*, 140–147.

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

Karresand, M., Warnqvist, A., Lindahl, D., Axelsson, S., & Dyrkolbotn, G. O. (2019). Creating a map of user data in ntfs to improve file carving. *IFIP International Conference on Digital Forensics*.

Lythgoe, J., & Partridge, J. (1989). Visual pigments and the acquisition of visual information. *Journal of Experimental Biology*, *146*(1), 1–20.

McDaniel, M., & Heydari, M. H. (2003). Content based file type detection algorithms. *36th Annual Hawaii International Conference on System Sciences*.

Memon, N., & Pal, A. (2006). Automated reassembly of file fragmented images using greedy algorithms. *IEEE transactions on image processing*, *15*(2), 385–393.

Microsoft. (2021a). Windows bitmapheaderinfo structure [Accessed: 2022-05-30].

Microsoft. (2021b). Windows metafile format [Accessed: 2022-05-30].

Nilsback, M.-E., & Zisserman, A. (2020). 17 category flower dataset [Accessed: 2021-04-10].

Nist, forensic images for file carving [Accessed: 2021-04-29]. (2021).

Pal, A., Shanmugasundaram, K., & Memon, N. (2003). Automated reassembly of fragmented images. *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, *4*.

Pal, A., Sencar, H. T., & Memon, N. (2008). Detecting file fragmentation point using sequential hypothesis testing. *Digital Investigation*, *5*.

Poisel, R., & Tjoa, S. (2013). A comprehensive literature review of file carving. *2013 International conference on availability, reliability and security*, 475–484.

Sari, S. A., & Mohamad, K. M. (2020). A review of graph theoretic and weightage techniques in file carving. *Journal of Physics: Conference Series*, *1529*(5), 052011.

Sencar, H. T., & Memon, N. (2009). Identification and recovery of jpeg files with missing fragments. *Digital Investigation*, *6*.

Shamos, M. I., & Hoey, D. (1975). Closest-point problems. *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*.

Tang, Y., Fang, J., Chow, K., Yiu, S., Xu, J., Feng, B., Li, Q., & Han, Q. (2016). Recovery of heavily fragmented jpeg files. *Digital Investigation*, *18*.

Tsamoura, E., & Pitas, I. (2009). Automatic color based reassembly of fragmented images and paintings. *IEEE Transactions on Image Processing*, *19*(3), 680–690.