

**Automated Reassembly of File Fragmented Images
Using Greedy Algorithms**

T H E S I S

for the Degree of

Master Of Science (Computer Science)

Anandabrata Pal

August 2005

**AUTOMATED REASSEMBLY OF FILE
FRAGMENTED IMAGES USING GREEDY
ALGORITHMS**

THESIS

Submitted in Partial Fulfillment
of the REQUIREMENTS for the

Degree of

MASTER OF SCIENCE (COMPUTER SCIENCE)

at the

POLYTECHNIC UNIVERSITY

by

Anandabrata Pal

August 2005

Advisor

Date

Department Head

Date

Copy No. ____

VITA

Anandabrata Pal was born in India on Nov 8, 1974. He received a B.S. degree in Computer Science from the New York Institute of Technology, New York, U.S.A., in 2001. He was a recipient of the Presidential Scholarship, and graduated summa cum laude. In addition he received the top awards for Computer Science, English and Extra-curricular activities on graduating. Since September 2002, he has been a Ph. D. candidate in the Computer Science Department in Polytechnic University, Brooklyn, NY, under the guidance of Prof. Nasir Memon. He currently has published 1 conference paper, IEEE journal paper awaiting publication, and has 1 patent pending. This is in addition to a paper published while working with Panasonic's Research and Development department in Princeton NJ. He has conducted research in the fields of video, image reassembly, and obfuscation.

ACKNOWLEDGEMENT

First, I would like to thank my thesis advisor, Professor Nasir Memon, for his invaluable support and guidance throughout the course of this Master's thesis. His extensive knowledge, enlightening direction, and continuous encouragement made my thesis work smooth and positive. He provided the guidance and knowledge that I lacked to complete this thesis.

I would also like to thank Kulesh Shanmugasundaram, who pioneered work in file fragmentation and who was a constant source of information and advice. His work in general file fragmentation provided me with a starting point in the research for my thesis. Finally, I would like to thank Pavel Jaromersky who provided a lot of help by reading and critiquing my work.

AN ABSTRACT

**AUTOMATED REASSEMBLY OF FILE
FRAGMENTED IMAGES
USING GREEDY ALGORITHMS**

by

Anandabrata Pal

Advisor: Nasir Memon

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science(Computer Science)

August 2005

The problem of restoring deleted files from a scattered set of fragments arises often in digital forensics. File fragmentation is a regular occurrence in hard disks, memory cards and other storage media. As a result, a forensic analyst examining a disk may encounter many fragments of deleted digital files, but is unable to determine the proper sequence of fragments to rebuild the files. To retrieve the maximum amount of information possible, fragmented deleted files will need to be recovered as well.

We investigated the specific case where digital images are heavily fragmented and there is no file table information by which a forensic analyst can ascertain the correct fragment order to reconstruct each image. The objective of this thesis is to identify techniques and algorithms to automate the process of image reassembly in the presence of fragmentation.

The image reassembly problem is formulated as a k - vertex disjoint graph problem and reassembly is then done by finding an optimal ordering of fragments.

We then provide techniques for comparing fragments and describe several algorithms for image reconstruction based on Greedy heuristics. An advanced image reassembly prototype called Automated Image Reassembler (*AIR*) was created to handle the reconstruction of fragmented images. This prototype provides experimental results showing that images can be reconstructed with high accuracy even when there are thousands of fragments and multiple images involved.

List of Figures

2.1	A simplified example of file fragmentation	5
3.1	(a), (b) and (c) show three different possibilities for the reconstruction of an image with 5 fragments.	8
3.2	A graph of seven fragments & two vertex disjoint paths (H_1CA & H_2BED)	10
4.1	Pixel values being compared when calculating candidate weights between two fragments	14
4.2	Example of reassembly using UP and NUP which can be rectified with the enhanced greedy based algorithms. Images from the mixed dataset.	17
4.3	Parallel Unique Path (PUP) algorithm example	20
6.1	Example of proper and improper reassemblies of the Caps image	30
6.2	Reassembly of FBI most wanted using different algorithms	31
6.3	Left: Original image of island from nature dataset. Middle and Right: Incorrect reconstructions of island pic showing large number of correctly ordered fragments	32

List of Tables

6.1	General dataset image information	30
6.2	Image Dataset Reassembly Information	34
6.3	Image Dataset Reassembly Information Continued	35
6.4	Best Reconstruction Algorithm for Datasets	35
6.5	Algorithm Percentage Reconstructions	35

Contents

List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Contributions of the Thesis	3
1.4 Organization of the Thesis	3
2 Fragmentation	4
2.1 Fragmentation Causes	4
2.2 File Allocation Tables	5
3 Statement of the Problem	7
3.1 Existing Fragmentation Research	7
3.2 Single Image Fragmentation	8
3.3 Multiple Image Fragmentation	9
4 Reassembly Techniques	12
4.1 Overview of Solution	12
4.2 Candidate Weight	12
4.3 Reconstruction Algorithms	15
4.3.1 Greedy Heuristic	15
4.3.2 Enhanced Greedy Heuristic	21
5 Automated Image Reassembler	24
5.1 Design Details	24
5.2 Splitting	24
5.3 Constructing Weight Tables	25
5.4 Reassembly Algorithms	26
6 Implementation and Experiments	28

7 Conclusion

36

Bibliography

37

Chapter 1

Introduction

1.1 Motivation

As technology evolves the number of people using computers, digital devices and the internet to commit criminal activities has increased [1]. Crimes include identity theft, illegal hacking of computers, and the distribution of child pornography. The increase in computer-related crime has caused law-enforcement agencies to seize digital evidence in the form of network logs, text documents, videos and images. In specific cases like those involving terrorism, the need to extract and analyze every possible bit of evidence becomes crucial. However, this digital evidence which is stored in the form of digital files can easily become fragmented and often requires reassembly to be useful.

File fragmentation is also a problem in the emerging field of digital photography recovery. Currently there are many packages that attempt to recover photos from digital storage devices, like flash cards and USB memory sticks, that have been damaged and/or accidentally formatted. While existing packages are able to recover most images if present, files that are fragmented are not usually recoverable.

File fragmentation normally is an unintended consequence of deletion, modification and creation of files in a storage device. Therefore, a forensic analyst investigating storage devices may come across many scattered fragments without any easy means of being able to reconstruct the original files. In addition, the analyst may not

easily be able to determine if a fragment belongs to a specific file or if the contents of the fragment are part of the contents from a particular file type (image, video, etc.).

The reconstruction of objects from a collection of randomly mixed fragments is a problem that arises in several applied disciplines, such as forensics [16], archaeology [2, 3], biology [4] and art restoration [5, 7]. In addition, the specific problem of jigsaw puzzle reassembly has also been studied extensively [8, 9, 10]. The digital forensic equivalent of the reconstruction of fragmented objects problem, which we call *reassembling fragmented documents*, however, has received little attention.

1.2 Problem Statement

Here we focus on the specific case of the problem, namely the reconstruction of images from a set of randomly ordered file fragments. We assume that the only information present is the actual contents of the fragments and that any file table records indicating a fragment's link to an image file and the order of fragments required to reconstruct the images is unavailable. In previous work, the reassembly of text files and binary executables from fragments was studied [16].

While we are focusing on the reassembly of fragmented images, rarely will analysts have fragments of only images available. The fragments themselves may be encrypted or compressed. Therefore, before the actual image reconstruction can occur some preprocessing may be needed to identify file types [14]. While certain compressed file types like JPEG may be hard to identify, identifying fragments that are clearly not image fragments (e.g. Text fragments) can also be beneficial. This is because we can then remove the identified non image fragments from the set of fragments to analyze.

In essence we are trying to solve the problem of having hundreds if not thousands of fragments with no immediate way to determine if a fragment belongs to a particular file and no direct way to determine the order of fragments required to reconstruct a file or set of files. For the purposes of our research we focused on images, more specifically Windows 24 bitmaps, however our research does translate to other image formats as well.

1.3 Contributions of the Thesis

Not much work has been done in the specific area of reconstruction of file fragments. In [16] first proposed solving the problem of reconstructing general files using novel techniques. By focusing on images specifically, we developed more robust techniques to compare fragments and further developed a set of algorithms to reconstruct the images. A prototype was built called AIR (Automated Image Reassembly) that attempted reassembly of images from multiple fragments using our developed techniques and algorithms. We show how even in cases where the image reassembly fails, it still helps a forensic analyst by providing enough correct fragments to reduce the complexity of the reassembly.

1.4 Organization of the Thesis

The remainder of the thesis is as follows. In the next chapter we provide background information into how files can get fragmented. In Chapter 3 we formulate the image reassembly problem in a more rigorous manner. In Chapter 4 we describe several approaches for a solution to the problem with a description of the algorithms and techniques used. Chapter 5 provides the design details of the Automated Image Reassembler prototype. Chapter 6 we provide experimental results from using AIR on different sets of fragmented images. Finally in Chapter 7 we present our conclusions with potential avenues for future research.

Chapter 2

Fragmentation

2.1 Fragmentation Causes

File fragmentation is an unavoidable problem that affects many computers using a variety of file systems. File systems such as Windows FAT, the UNIX Fast File System and highly active file systems, like that of a busy database server, will often fragment files into discontinuous blocks. Typically a hard disk is broken into clusters of equal size, for example hard-drives formatted with FAT32 clusters can be 4K each. When a file larger than the cluster size is saved to disk, it will occupy more than one cluster. Fragmentation can occur when the file system cannot find sufficient contiguous clusters for a file. File extension is another source of fragmentation. If a file is extended/appended to, and there is no room at the end to grow it contiguously, the file will be fragmented. Finally, deleting files may partition the free space which could result in further fragmentation.

Fig. 2.1 is a very simple example of a disk with 10 clusters. Fig. 2.1 (a) displays 4 image files A, B, C, and D which are stored in consecutive clusters. This means there is no fragmentation. In Fig. 2.1 (b) the image B was deleted and the clusters that B originally occupied are marked as empty. Next a user wants to save an image E that requires 3 clusters to store. The OS then saves the file in clusters 4, 5 and 10, as shown in Fig. 2.1 (c), thus leading to fragmentation. The OS maintains a data structure called a file table where it records the sequence of clusters that is needed to retrieve all stored files.

1	2	3	4	5	6	7	8	9	10
A	A	A	B	B	C	C	D	D	-

(a) Unfragmented disk with 10 clusters (Cluster 10 has no data)

A	A	A	-	-	C	C	D	D	-
---	---	---	---	---	---	---	---	---	---

(b) Free space fragmentation after File B is deleted

A	A	A	E	E	C	C	D	D	E
---	---	---	---	---	---	---	---	---	---

(c) Fragmentation of File E

Figure 2.1: A simplified example of file fragmentation

When deleting files an OS (e.g. Windows) will typically not delete the contents of the file, but will instead mark the clusters of the storage as free/available and will delete the file table entry of the file only. Therefore, though file table information for an image may no longer be available, the actual image contents may still be present. So if the user then deletes image E and an analyst analyzes the hard-disk, he may be able to retrieve the information in clusters 4, 5, and 10 with no way to determine the proper sequence to reassemble the original image.

2.2 File Allocation Tables

In Windows FAT32 the file table is known as the File Allocation Table (FAT). Every cluster in the disk has an entry in the FAT. Every entry for a cluster in the FAT contains a value indicating if it is a free cluster, a reserved cluster, a bad cluster, the last cluster of a file or the next cluster for the file. If a cluster belongs to a file spanning multiple clusters, then it will point to the next cluster containing the file data (unless it is the last cluster, in which case it will indicate the fact).

In the real world file creation, deletion, and modification occurs frequently

resulting in fragmentation. Most hard drives now can store gigabytes of files and a 10GB hard drive having 4K clusters would have more than 2.6 million clusters. A typical high resolution image, depending on the format used, can utilize anywhere from a few Kilobytes to a few hundred Megabytes of storage. Therefore, most images will be saved in multiple clusters on a storage device. As a result, a forensic analyst investigating a storage disk (i.e. a hard disk), may find hundreds to thousands of disk clusters that correspond to fragments of previously deleted images. Without adequate file table information (caused by deletions, formatting or corruption) it is difficult to put the fragments back together in their original order. Similarly, a lot of memory cards, media sticks and other storage media for digital cameras typically store files using the FAT32 system which is subject to heavy fragmentation. There are many commercial software packages, like PhotoRescue and MediaRECOVER Image Recovery that attempt to recover deleted images from hard disks, memory sticks, compact flashes and other devices. However, in the presence of fragmentation these programs create only partial or incorrect image reconstructions. As far as we are aware there has been no published literature on the reassembly of fragmented images. The next chapter introduces the reassembly problem formally, and describes our approaches to solve the problem.

Chapter 3

Statement of the Problem

3.1 Existing Fragmentation Research

The problem of reassembly of image fragments differs slightly from the reassembly of fragments like shards of pottery or jigsaw puzzles. First the sizes of all the fragments in our problem will be the same, this is because the fragments correspond to disk clusters that are normally fixed in size on storage devices. File fragments also do not have a set shape as they are simply consecutive bytes of a file stored in a disk. Therefore, we are not able to use shape matching [11, 15] to reconstruct fragmented images. In fact, the shape of the fragment is dependent on the position that it is in the image. For example, Fig. 3.1, shows three potential reassembly sequences of an image consisting of 5 fragments. The figure shows the size of each fragment to be the same (8 pixels) and the shape of the fragment is shown clearly to be dependent on where the fragment is being used for reconstruction. Finally, the fragments in the reassembly of physical objects and jigsaw puzzles may potentially link to multiple fragments, while our fragments will typically be connected to at most two other fragments (one above and one below). This is because we assume that each fragment will contain at least a width amount of pixels.

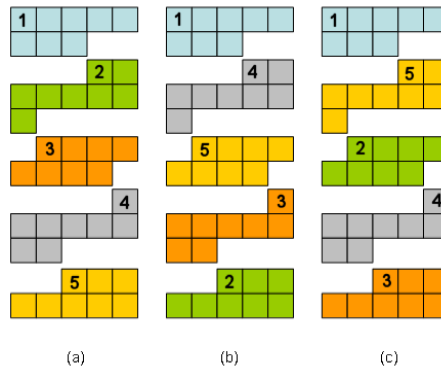


Figure 3.1: (a), (b) and (c) show three different possibilities for the reconstruction of an image with 5 fragments.

3.2 Single Image Fragmentation

We begin with evaluating the case of a single image split into multiple fragments. Suppose we have a set $\{A_0, A_1, \dots, A_n\}$ of fragments of an image A . We would like to compute a permutation π such that $A = A_{\pi(0)} || A_{\pi(1)} || \dots || A_{\pi(n)}$, where $||$ denotes the concatenation operator. In other words, we would like to determine the order in which fragments A_i need to be concatenated to yield the original image A . We assume fragments are recovered without loss of data and no fragments are missing or corrupted. That is, we assume concatenation of fragments in the proper order yields the original image intact.

Note that in order to determine the correct fragment reordering, we need to identify fragment pairs that are adjacent in the original image. To quantify the likelihood of adjacency one may assign *candidate weights* $C_{(i,j)}$, representing the likelihood that fragment A_j follows A_i . There are various techniques that can be used to calculate these weights. For example, when dealing with image fragments these weights can be computed based on gradient analysis across the boundaries of each pair of fragments. Once these weights are assigned, the permutation of the fragments that leads to correct reassembly, among all possible permutations, is likely to maximize (or minimize) the sum of candidate weights of adjacent fragments. This observation gives us a technique to identify the correct reassembly with high probability. That

is, we want to compute the permutation π such that the value

$$T = \sum_{i=0}^{n-1} C_{(\pi(i), \pi(i+1))} \quad (3.1)$$

is maximized (if a greater weight implies a better match) or minimized (if a lower weight implies a better match) over all possible permutations π of degree n .

The problem of finding a permutation that maximizes (or minimizes) the sum in equation (1) can also be abstracted as a graph problem if we take the set of all candidate weights (C) to form an adjacency matrix of a complete graph of n vertices, where vertex i represents fragment i and the edge weight e_{ij} represent the likelihood of fragment j following fragment i . The proper sequence π is a path in this graph that traverses all the nodes and maximizes (or minimizes) the sum of candidate weights along that path. The problem of finding this path is equivalent to finding a maximum weight Hamiltonian path in a complete graph and the optimum solution to the problem turns out to be intractable[12].

3.3 Multiple Image Fragmentation

While the single image fragmentation case is important to consider, it is not that realistic. More often than not multiple images each with multiple fragments will have to be reassembled. The complexity of the problem increases because unlike the single image case, we cannot tell with certainty if a fragment was part of a particular image. In addition, the resolutions and, as a result, the number of fragments to reconstruct each image may vary.

When analyzing k images fragmented into n total fragments, we can represent the fragments as a complete graph of n vertices, where each edge has a cost/weight equivalent to the candidate weights between the equivalent fragments. Assuming P_i is the path or sequence of fragments reassembled for the i th image, then the correct solutions $(P_1, P_2 \dots, P_k)$ are k vertex disjoint paths in this graph. The problem of finding the optimal paths is equivalent to finding k vertex disjoint paths in a complete graph (see Fig. 3.2) such that the sum of all the paths is maximized (or

minimized) and the optimum solution to the problem turns out to be NP-Complete [18].

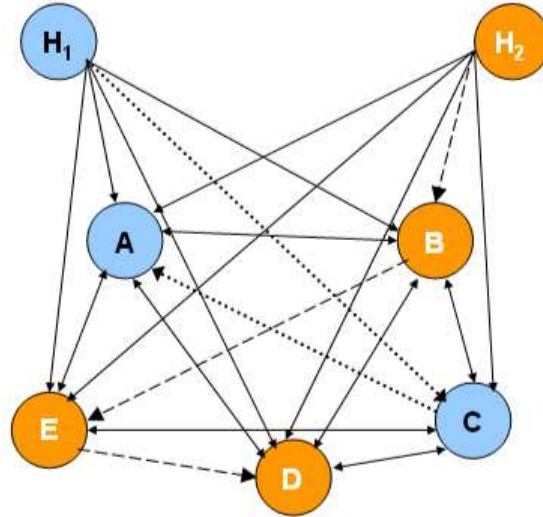


Figure 3.2: A graph of seven fragments & two vertex disjoint paths (H_1CA & H_2BED)

In practice the problem is made slightly easier because we can identify the starting fragments from each image. This information is determined by analyzing each fragment and trying to determine if the fragment is an image header. Most image types have specific header formats to identify the type as well as image specific information. For example, every Windows BMP image begins with a 54-byte image header. All BMPs are required to start with the characters 'BM'. By identifying all the fragments that are image headers, we can determine the type of image, the number of images to reconstruct and the starting fragment for each image. In addition, with information obtained from the header, we can determine the resolution of an image and the size of the image. From the resolution we are able to determine the width of the image which is equal to the number of pixels used for fragment candidate weight calculations. Fig. 4.1 shows a simplified example of matching two fragments for an image of width equal to 5 pixels. From each image's size we are able to identify the total number of fragments required to build the original image.

So we have now defined the image reassembly problem as a k -vertex disjoint path problem. We can identify the image header fragments and as a result we are able to identify the number of images and the number of fragments required to reconstruct each image. Also the header provides information of each image's resolution and this enables us to compare fragments and evaluate candidate weights. The candidate weights can now be used to determine the optimal paths for reassembly and we present techniques to evaluate these weights in the next chapter.

Chapter 4

Reassembly Techniques

4.1 Overview of Solution

Once we can identify the header and number of fragments required to reconstruct an image we need a technique to calculate the likelihood that a fragment follows another fragment in the reconstruction path. Using this information we can then apply a variety of algorithms to attempt to reconstruct the original image(s). Clearly the techniques used to determine the likelihood that a fragment follows another should be based on the actual image information and pixel data.

4.2 Candidate Weight

In order to recreate a path for reconstruction of images, we need to be able to assign weights between all fragments indicating the likelihood that a fragment follows another fragment. We call this weight the Candidate weight, and to be more precise the value for the candidate weight of fragments $C_{(i,j)}$ indicate the value that fragment j follows fragment i .

In this section we present three different techniques to evaluate the candidate weights between any two fragments. Descriptions of the fragment comparison methods for Pixel Matching (PM), Sum of Differences (SoD) and Median Edge Detector (MED) are presented.

When comparing any two fragments i and j , the candidate weight for $C_{(i,j)}$ involves comparing the last w (image width) pixels of fragment i with the starting w pixels of fragment j . Since each fragment typically contains more than w pixels, the weights for $C_{(i,j)}$ will normally be different than those for $C_{(j,i)}$, since different pixels will be compared for the two comparisons. Intuitively, this makes sense since if fragment j follows fragment i in an image then the value assigned to $C_{(i,j)}$ should be different (and better) than the value assigned to $C_{(j,i)}$.

Assigning weights becomes slightly more complicated when images with different widths are fragmented, as multiple weights will have to be calculated between any two fragments. This is because the number of pixels used in the calculation will be different for each width and as a result the weights computed may differ. As we are initially unable to determine which image a fragment belongs to, we need to compute weights between fragments i and j for every unique width encountered in the k image header fragments.

The basic approach for assigning candidate weights for a pair of fragments essentially involves examining pixel gradients that straddle the boundary formed when the fragments are joined together. One relatively simple technique that can be used is to compute the absolute sum of prediction errors for the pixels along the boundary formed between the two fragments (Fig. 4.1). That is, prediction errors are computed for pixels in the last row of the first fragment and the pixels in the first row of the second fragment. It is also known that an image consists mostly of smooth regions and the edges present have a structure that can often be captured by simple linear predictive techniques. Hence another way to assess the likelihood that two image fragments are indeed adjacent in the original image is to compute prediction errors based on some simple linear predictive techniques. Examples of such techniques are those used in lossless JPEG, or even better, the MED predictor used in JPEG-LS [13]. We compared the results of 3 techniques to determine the prediction errors between two fragments:

1. *Pixel Matching (PM)*: This is the simplest technique whereby the total number of pixels matching along the edges of size w for the two fragments are summed. In Fig. 4.1 the width (w) is 5 and PM would compare to see if each numbered pixel

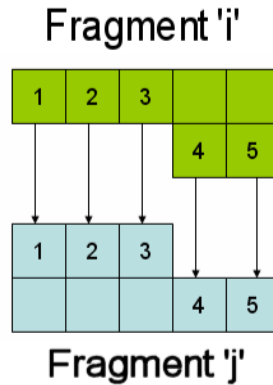


Figure 4.1: Pixel values being compared when calculating candidate weights between two fragments

in fragment i matched in value with the same numbered pixel in fragment j . If the pixels matched the value of the PM weight would be incremented by one. For PM, the higher the weight the better the assumed match.

2. *Sum Of Differences (SoD)*: The sum of differences is calculated across the RGB pixel values of the edge of every fragment with every other fragment. In Fig. 4.1 SoD would sum the absolute value of the difference between each numbered pixel in fragment i with the same numbered pixel in fragment j . For SoD, the lower the weight the better the assumed match.

3. *Median Edge Detection (MED)*: Each pixel is predicted from the value of the pixel above, to the left and left diagonal to it [13]. Using MED we would sum the absolute value of the difference between the predicted value in fragment j and the actual value. When MED was used, the lower the weight the better the assumed match.

Experimentally, we found the SoD and MED techniques to be far more useful than the PM technique. We create weight tables storing the values of $C_{(i,j)}$ for every i and j . Now that we have techniques to compare and weight a pair of fragments, we now need to use the weights as inputs into algorithms to reconstruct the images. In the following chapter, we describe these algorithms.

4.3 Reconstruction Algorithms

We are now ready to present algorithms to reconstruct the images based on the candidate weights between fragments. This section describes 8 different algorithms used in the reassembly of images. The algorithms are classified by their ability to create vertex disjoint paths or not, whether or not images are reconstructed serially or in parallel, and according to the heuristic used (Greedy or Enhanced Greedy).

Edge and vertex disjoint path problems occur commonly in VLSI, scheduling and networking [24, 23]. The use of Greedy approximation algorithms to solve edge and vertex disjoint problems has been studied extensively [21, 20, 22]. However, in this paper some of the algorithms presented do not necessarily lead to disjoint paths. The algorithms presented that create vertex disjoint paths are called unique path (UP) algorithms (i.e. each fragment is assigned to one and only one image). The problem with UP algorithms is that a fragment assigned incorrectly to image A, but belonging to image B will always result in A and B reconstructing incorrectly. Therefore, we also present non-unique path (NUP) algorithms (a fragment may be used more than once for image reconstruction). While NUP algorithms may solve the problem of error propagation in UP algorithms, a fragment may be reused in the reconstruction of one or more images. If a fragment is reused in more than one image then clearly there was an error in the reconstruction of one of the images. The algorithms can also be classified as sequential or simultaneous algorithms. The sequential algorithms reconstruct a single image in its entirety before moving on to the next image. The simultaneous algorithms try to reconstruct all k images in parallel.

All the algorithms presented use one of two heuristics. The greedy heuristic and our variation of the greedy heuristic called Enhanced Greedy.

4.3.1 Greedy Heuristic

Starting with the header fragment, the greedy heuristic reconstructs an image by choosing the best available fragment, selecting it for reassembly, and then choosing this fragment's best available match. This process is repeated until the

image is reconstructed. The header fragment is stored as the first fragment in the reconstruction path P of the image and then it is set as the current fragment s . After selecting a fragment s , the fragment's best successor match t is chosen. The best match is based on the best candidate weight as determined by one of the three weight calculation techniques provided earlier. t is then added to the reconstruction path P and becomes the new current fragment s . This process is repeated until the image is reconstructed.

```
function greedy(currentFragment, availableFragments[]){
    for(x= 1;x < number of availableFragments[]; ++x) {
        set bestMatch = get best < x > fragment for currentFragment
        if (bestMatch found in availableFragments[])
            return bestMatch
    }
}
```

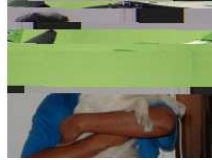
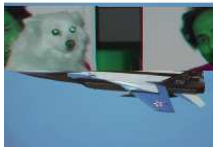
The candidate weights used by the various greedy algorithms to determine best matches are sorted. For every fragment we sort its weights with the other $n - 1$ fragment which takes $O(n \log n)$ steps.

Prior to running any of the algorithms based on the greedy heuristic, we calculate the candidate weights between all fragments. For n fragments this takes $O(n^2)$ steps. For every fragment i we then sort the other $n - 1$ fragment numbers based on candidate weights which takes $O(n \log n)$ steps. However, we have to sort for every fragment so the complexity will be $O(n^2 \log n)$ for the sorting step. We now present 4 algorithms based on the Greedy heuristic.

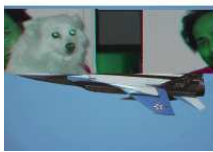
1. **Greedy SUP:** Greedy Sequential Unique Path is a sequential algorithm using the greedy heuristic. When the algorithm assigns a fragment to an image reconstruction, the fragment will be unavailable for selection in the reconstruction of any other images. Though this creates vertex disjoint paths, the problem is that the paths depend on the order of images being processed. More specifically



(a) Proper reassembly of images with Greedy UP when processing dog first



(b) Improper reassembly of images with Greedy UP when processing plane first



(c) Improperly Reassembled image of Plane with Greedy NUP

Figure 4.2: Example of reassembly using UP and NUP which can be rectified with the enhanced greedy based algorithms. Images from the mixed dataset.

the algorithm proceeds as follows.

We randomly choose any order for processing the images, however, changing the order may result in different reassembly results. Let P_i be the reconstruction path of image i and the header fragment for i be identified as h_i . To start we choose the header h_i as the first fragment in the reconstruction path (i.e. assign $P_i = h_i$). We set the current fragment equal to the header, $s = h_i$, and then find s 's best available greedy match t . The best available match is s 's best match t that has not been used in any another image reassembly. We put t in the reconstruction path ($P_i = P_i || t$) and then set it as the current fragment for processing ($s = t$). We then find its best match and repeat until the image is reconstructed. We then proceed to the next image and repeat the process until all k images have been reassembled.

Looking at Fig. 4.2, with Greedy SUP, both the image of the dog and plane will reconstruct perfectly if the dog is reconstructed first and then the plane 4.2(a). If the plane is reconstructed first it will reconstruct incorrectly thus

causing the dog to reconstruct incorrectly 4.2(b). This is because some of the fragments of the dog will be assigned to the plane, and then the dog will reconstruct incorrectly because those fragments of the dog assigned to the plane will not be available. Therefore, since image reconstruction may be dependent on the order of images being processed for reassembly, we do not present results for this algorithm.

As fragments are already sorted by matches the time taken to find the best match is constant. Therefore, to reassemble n fragments will take $O(n)$ time. Taking into account the preprocessing steps of calculating the weights and sorting, the complexity of the algorithm is $O(n) + O(n^2) + O(n^2 \log n) = O(n^2 \log n)$.

2. **Greedy NUP:** Greedy Non Unique Path is also a sequential algorithm using the greedy heuristic. Since this is a NUP algorithm any fragment, other than a header fragment, that was chosen in the reconstruction of an image will be available for selection in the reassembly of another image. This prevents errors from propagating but as mentioned earlier, does not necessarily lead to disjoint paths.

Unlike Greedy SUP, here different orders for processing the images will never lead to different reassembly results. Therefore, we can randomly choose any image header to start with. We then choose this header h_i as the first fragment in the reconstruction path (i.e. assign $P_i = h_i$). We set the current fragment equal to the header, $s = h_i$, and then find s 's best greedy match t . Even if the best match was used in another reassembly we still select it and put it in the reconstruction path ($P_i = P_i || t$) and then set it as the current fragment for processing ($s = t$). We then find its best match and repeat until the image is reconstructed. We then proceed to the next image and repeat the process until all k images have been reassembled.

Looking at Fig. 4.2(c) we can see that because Greedy NUP can choose fragments that were chosen in an earlier image, mistakes like that of the dog not reconstructing even though the plane was reconstructed incorrectly (by using some of the fragments from the dog) do not occur. It can be seen clearly that

some of the fragments from the dog are used in both reconstructions. Greedy NUPs complexity is the same as Greedy SUP $O(n^2 \log n)$.

3. **Greedy PUP:** Greedy Parallel Unique Path creates UP reconstructions without having the reconstructions depending on the order of the images being reconstructed. It is a variation of Dijkstra's single source shortest path algorithm [19], which we use to reassemble images simultaneously. Starting with the image headers we choose the best match for each header, pick the header-fragment pair with the best of all the best matches and assign that fragment to the header. We then repeat the process until all images are reconstructed.

More formally, we store the k image headers as the starting fragments in the reconstruction paths P_i for each of the k images. We maintain a set $S = (s_1, s_2, \dots, s_k)$ of current fragments for processing where s_i is the current fragment for the i th image. Initially all the k starting header fragments are stored as the current fragments for each image (i.e. $s_i = h_i$). We then find the best greedy match for each of the k starting fragments and store them in the set $T = (t_1, t_2, \dots, t_k)$ where t_i represents the best match for s_i . From the set T of best matches the fragment with the overall best weight is chosen. Let us assume that this fragment was t_i . Then we add the fragment to the reconstruction path of the i th image ($P_i = P_i || t_i$), replace the current fragment for the i th image ($s_i = t_i$) and evaluate the new set T of best matches for S . We then again find the best weight among the fragments in T , and repeat the process until all images have been reconstructed.

Fig. 4.3 shows an example of the algorithm where there are three images being reconstructed. Fig. 4.3(a) shows the headers H_1, H_2 and H_3 of the three images and their best matches. The best of all the matches is presented with a dotted line and is the H_2 -6 pair of fragments. Fig. 4.3(b) now shows the new set of best matches after fragment 6 has been added to the reconstruction path of H_2 . Now fragment 4 is chosen once each for fragments 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 fragment 6 is determined 4.3(c). This process continues

until all images are reconstructed.

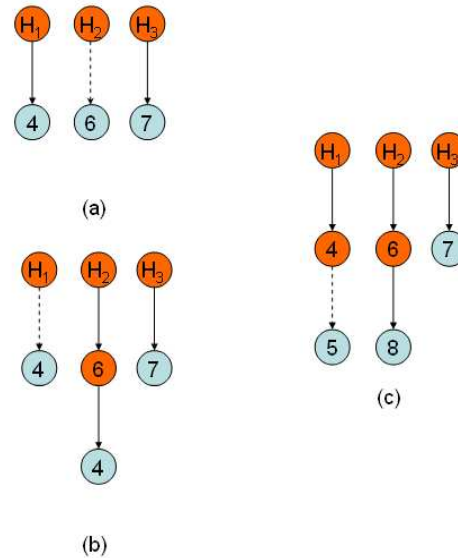


Figure 4.3: Parallel Unique Path (PUP) algorithm example

The problem with Greedy PUP is that the best fragments being matched with the current set S of fragments may be better matches for fragments that have not been processed as yet, thus leading to error propagation again. For example fragment 4 may have been a better match for a fragment that was not processed as yet, and having it chosen would lead to errors in the reconstruction of at least H_1 . Again finding the best match takes constant time and, therefore, the complexity of the algorithm is dependent on the preprocessing steps and is equal to $O(n^2 \log n)$.

4. **Greedy SPF UP:** The advantage of the NUP algorithms is that error propagation does not occur from assigning a fragment to an image it did not belong to. However, the disadvantage is that we know there will be an error if a fragment is used more than once in the reconstruction of images. Greedy Shortest Path First is an attempt to get the benefits of the NUP algorithms, while being an UP algorithm. Greedy SPF is a UP algorithm that is a variation of the Shortest-Path-First Greedy algorithm [20]. Here the assumption made is that

the best reconstructed image is the one with the lowest average path cost. In our algorithm, we run the Greedy NUP across all the images computing the total cost for each image. The total cost is the simply the sum of the weights between the fragments of the reconstructed image. Note: we use the NUP and not SUP so a fragment may be assigned to one or more images to prevent error propagation for an incorrectly assigned fragment. Since images may use different number of fragments, we divide the total cost of an image by the number of fragments in the image to get the average cost of each path. We then select the image with the reconstruction path having the lowest average cost, mark it as reassembled and remove its fragments from the set of available fragments. We then redo the Greedy NUP on the remaining images and again remove the fragments of the path with the lowest average cost and repeat until all images have been reconstructed. It takes n calculations for the first iteration, $n - n_1$ for the second and so on. To be more precise Greedy SPF UP takes $\sum_{i=1}^k (n - n_i)$ operations which in the worst case is $O(n^2)$. The complexity however is still dominated by the preprocessing steps of sorting and is equal to $O(n^2 \log n)$.

4.3.2 Enhanced Greedy Heuristic

The greatest problem that the Greedy heuristic has is that it chooses the best available matching fragment t for the current fragment s without attempting to take into account the possibility that fragment t may be an even better match for another fragment that has not been processed as yet. The Enhanced Greedy heuristic attempts to address this problem. Intuitively, it attempts to determine if the best match for a fragment may be an even better match for another fragment. If this is the case then the next best match is chosen, and the process is repeated until a fragment is found that is not a better match for any other fragment.

Just like the greedy heuristic, the Enhanced Greedy initially chooses the best available fragment t for the current fragment s being processed. t 's best predecessor fragment b is then checked. Here b is the fragment that has the best candidate weight when being compared to t , however, t may not necessarily be the best match for b . If

$b = s$, the current fragment matches better than all other fragments with t , then the fragment t is selected in the reconstruction. If $b \neq s$ then b 's best match is checked. If this is equal to t (t was a better match for b and b 's best match was t), then the next best child match for s is determined and evaluated as before.

```

function enhancedGreedy(currentFragment, availableFragments[]) {
  for(x= 1;x < number of availableFragments[]; ++x) {
    set bestMatch = get best < x > fragment for currentFragment

    if (bestMatch found in availableFragments[]) {
      set bestPredecessor = get best predecessor match for bestMatch
      if(bestPredecessor = currentFragment)
        set bestFinalMatch = bestMatch
      else {
        set bestPredecessorBestMatch = get best match for bestPredecessor
        if (bestPredecessorBestMatch = bestMatch)
          Repeat loop for next best match
        else {
          if (currentFragment is > 3rd best fragment
            for bestMatch)
            Repeat loop for next best match
          else
            set bestFinalMatch = bestMatch
        }
      }
    }
  }
  return bestFinalMatch
}
}
}

```

All the algorithms presented using the enhanced greedy heuristic are similar

to those presented earlier with the exception of using the enhanced greedy heuristic in place of the greedy heuristic for determining best fragment matches. As with the greedy heuristic, candidate weights are precomputed and sorted. However, in the enhanced greedy heuristic we also sort the best predecessor fragment matches for each fragment. This additional step takes $O(n^2 \log n)$ as well and for all the algorithms the sorting still dominates the complexity. Using the enhanced greedy heuristic we get four more algorithms: 5) **Enhanced Greedy NUP**, 6) **Enhanced Greedy SUP**, 7) **Enhanced Greedy PUP** and 8) **Enhanced Greedy SPF UP**.

Now we have all the pieces required to describe our approach to reassembling images given a collection of their fragments. We first examine all fragments and identify those that correspond to image headers having known formats. From these, the number of fragmented images and the width w of each fragmented image is determined. We then compute weights that represent the likelihood of adjacency for a given pair of fragments by computing the sum of absolute prediction errors across the ending w pixels of the first fragment to the starting w pixels of the second fragment. Repeating the process for all fragments results in a complete weighted and directed graph. We then use the various algorithms presented as a solution for computing maximum (or minimum) weight disjoint paths to attempt to reassemble the images correctly. The actual reordering is likely to be contained in this set or at worst can be easily derived from this set by a forensic analyst. We are now ready to present the experimental results from running the algorithms described.

Chapter 5

Automated Image Reassembler

5.1 Design Details

The Automated Image Reassembler (*AIR*) is designed to showcase the effectiveness of the techniques proposed in reassembling fragmented images. All three of the fragment comparison techniques as well as all the algorithms presented have been implemented on the system. In addition, *AIR* will split files/images into random fragments, and note the fragment order required for correct reassembly. Using this information the program can determine if an image was reconstructed correctly. In a real world system, this part would have to be done by a human. Finally, all reconstructions could be saved as windows bitmaps for later analysis and presentation. All reconstruction images in this thesis were created by *AIR*.

5.2 Splitting

Each of the image datasets provided in the next chapter, were split into random 4K fragments. Currently, *AIR* only supports Windows 24-bit Bitmap images, however, the techniques carry over to almost all non-progressive image formats. It should be noted that 24-bit map images are built from the bottom up, and therefore, the first fragment contains both header information and the pixel information of the first pixels at the bottom of the image. 4K fragments are used because they are common FAT32 size clusters that are quite often used in digital media. Again, as

long as the fragments are large enough to store at least a width of pixel information our techniques work. The actual splitting operation attempts to randomly assign numbers to each fragment, thus indicating a worst case scenario where fragments are randomly distributed on a disk. It should be noted that our techniques were designed to work in this worst case scenario, even though randomly distributed fragments will very rarely occur in a real system. Finally, when splitting images to simulate randomly distributed fragments, *AIR* will store the sequence of fragments required to reconstruct the image. This information is never used in the actual reassembly process, but used to easily verify when an image has been reconstructed correctly and was also used to determine at which fragment a reassembly algorithm failed.

5.3 Constructing Weight Tables

Once a set of fragments is loaded, *AIR* tries to determine if weight tables already exist for the fragments. If it does not, *AIR* goes through the following sequence of procedures:

1. **Identify Headers:** As mentioned earlier *AIR* scans each fragment to determine if it is a header. It does this by looking for a unique Windows BMP signature at the start of the fragment. The number of headers found equals the number of images that need to be reconstructed.
2. **Identify Number of Fragments:** For each header identified, *AIR* then attempts to determine the number of fragments required to reconstruct the image. Windows 24-bit bitmaps use 3 bytes to store the RGB value of each pixel. In addition the size of the header is fixed, and the size of each fragment is 4096 bytes. Therefore by multiplying the height and the width (both values present in the header) one can determine the number of pixels. The total size of the image in bytes then becomes the size of the header plus the number of pixels multiplied by 3. We then simply divide this by 4096 to determine the number of fragments required to reconstruct that header.

3. **Create Weight Tables:** After a user chooses which of the three comparison techniques he/she wishes to apply (SoD, PM or MED), *AIR* attempts to create weight tables comparing each available fragment with every other fragment. A weight table is essentially a table where the rows indicated the fragment i and the columns the fragment j so the value in the cell $ij = C_{ij}$. In addition each weight table stores an additional table that contains the a list of sorted best matches for each fragment. This enables algorithms to quickly determine the best match during reassembly.

In addition, the weight of a fragment with itself is considered to be undefined (obviously a fragment cannot be attached to itself in an image) so $C_{(i,i)} = \#undefined\#$. Next the weight between a fragment i and fragment j where j has been identified as a header will also be $\#undefined\#$. Finally, the weight $C_{(i,j)} = \#undefined\#$ in all cases where i is not a header fragment and i is the starting fragment of the image.

Finally, it should be noted that for every unique width encountered weight tables must be made. This is because the weights are calculated across a width of pixels and different widths will result in different values for the weight. Finally, since pixel information is stored in three bytes and it is impossible to initially determine if a fragment (unless it is a header fragment) started at the B, G or R value in the original image we have to make 3 weight tables for each width to account for all possibilities.

5.4 Reassembly Algorithms

Finally, the user can choose from one of the eight available greedy and enhanced greedy algorithms. Each image reconstruction (correct and incorrect) is displayed on the screen along with the sequence of fragments that resulted in the reconstruction and the sequence of fragments required to correctly reconstruct the image. A simple checkbox displays if the image was reconstructed correctly or not. Fragments from images that have been reconstructed correctly can be marked as belonging to that image and discarded. Thus any of the algorithms can be run again

on fewer fragments in an attempt to rebuild images that were not reconstructed in earlier attempts. Finally for any reconstruction the user can choose a fragment and swap it with another fragment from a sorted list of best matches and reconstruct the image from that point on using greedy or enhanced greedy techniques. The one functionality not built in but that would be required in any commercial version of this application is the ability for the user to identify correctly ordered blocks of fragments that could then be used in further iterations of the algorithms to enhance the reassembly process. The next chapter details some results from our experiments using *AIR*.

Chapter 6

Implementation and Experiments

This section presents experimental results and discussion of the results. We used 24-bit color Windows bitmaps as the images in our experiments. Seven datasets were chosen for experiments. All pictures used were saved or converted into 24-bit color bitmaps. The pictures within a dataset were then randomly fragmented together into 4K (4096 Byte) sizes. 4K sizes were used because it is a commonly used size of FAT32 clusters. Simple header checking code was able to determine the headers for each image in a dataset.

In our experiments we also assumed that no fragments are missing and that spurious fragments (fragments that are not part of any image, like text fragments) are not present. However, in a realistic file reconstruction scenario, both missing and spurious fragments will be evident. In the case of missing fragments some images may not be reconstructed correctly, however, again a large number of fragments that are correctly ordered will be present and a forensic analyst will be able to identify these fragments and use that information to redo the reconstructions. When spurious fragments are present, they will likely match very poorly during our fragment candidate weight calculations. So spurious fragments will typically not affect the results of our algorithms. Finally, if a file consists of multiple fragments which are stored in multiple clusters, then typically the first fragment of the file will be stored in a cluster number lower than the next fragment, the second fragment will be stored in a cluster higher than the first but lower than the third and so on. We can utilize this fact also to enhance the recovery process by ignoring fragments stored in clusters

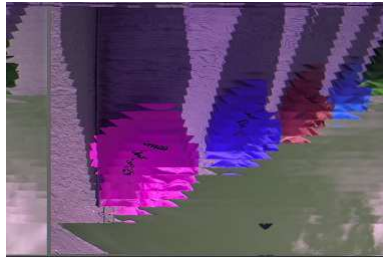
numbers lower than the last cluster used in the reconstruction of a file. However, our experiments were done assuming that a fragment could be stored anywhere on the disk (our fragmentation was completely random) and we still got excellent results.

In our experiments we found the best values for edge weights to be derived from the SoD prediction scheme. So for the datasets presented only the SoD technique was used for all algorithms. Once an algorithm was chosen and a set of images reassembled, the fragments for the properly reconstructed images were ignored in an attempt to correctly reconstruct the other images. No more than 3 iterations were needed to reconstruct the majority of images in almost all cases. Table 1 shows the general image statistics for the 7 datasets, and Table 2 shows the number of images reconstructed using each algorithm. Table 3 shows the best algorithms for reassembling each dataset on the first iteration. Finally Table 4 shows the percentage of images reconstructed by each algorithm. Results for the UP algorithms were not shown because they rely on the order of reconstruction.

The first data set was a single image 768x512 of caps hanging on a wall. The algorithms based on the regular greedy heuristic fail to correctly reconstruct the image as the border results in incorrect reassembly. The algorithms using enhanced greedy as the underlying basis are able to catch this issue and reconstruct the image in its entirety. Note the Greedy NUP, PUP and Greedy SPF all behave similarly as only one image was broken into fragments (Fig. 6.1).

The second data set used was a collection of 4 512x512 images from the USC-SIPI Image Database. The images used were Girl (Lena), Baboon, Airplane (F-16) and Sailboat on lake. The two PUP algorithms resulted in perfect reconstruction of all 4 images. All other algorithms resulted in near perfect results with the only discrepancy occurring with the last fragments in Baboon and Sailboat being swapped, this was not even visually obvious.

The third data set used was a collection of 24 facial images of the Indian and Australian cricket teams. The Enhanced greedy SPF and Greedy SPF algorithm resulted in 22 of the 24 being reconstructed perfectly. All other algorithms resulted in 21 perfect reconstructions. Again the 2 or 3 images not reconstructed perfectly look visually to be perfect, the last fragments containing hair standing up were swapped.



(a) Caps Image with fragments out of order.



(b) Same image with fragments in correct order.

Figure 6.1: Example of proper and improper reassemblies of the Caps image

The fourth data set that was used consisted of 10 images retrieved from the FBI's most wanted website (Fig. 6.2). The various algorithms worked with different levels of accuracy. The best algorithm was the enhanced greedy SPF algorithm which resulted in all 10 images being reconstructed.

The fifth data set used was a collection of 5 images of fighter planes. Here all the algorithms other than the standard Greedy NUP result in full reconstruction of all 5 of the images.

The sixth data set used was a collection of 5 nature scene images. Only 1

Table 6.1: General dataset image information

<i>Dataset</i>	<i>Images</i>	<i>Resolution</i>	<i>Fragments</i>
Caps On The Wall	1	768x512	289
USC-SIPI Images	4	512x512	772
Cricketers (Faces)	24	100x150	264
FBI Most Wanted	10	152x203, varied	265
Aircraft	5	152x203, 512x512	282
Nature	5	varied	320
Mixed	10	varied	389



Figure 6.2: Reassembly of FBI most wanted using different algorithms

of the pictures could be reconstructed correctly with the normal Greedy and PUP algorithms. All other algorithms failed, however, large blocks of correctly ordered fragments did appear and could be used to reconstruct the images (Fig. 6.3).



Figure 6.3: Left: Original image of island from nature dataset. Middle and Right: Incorrect reconstructions of island pic showing large number of correctly ordered fragments

The final set used was a collection of 10 relatively non-related images. There were 4 images of fighter planes mixed in with 1 image of an actress posing, 1 image of a woman's face, an image with a dog, an image of a beach, a digital drawing of star trek and a small banner from the web. All the algorithms other than Greedy NUP resulted in perfect reconstruction of all 10 images.

From the results obtained so far, we can conclude that the Enhanced Greedy SPF seems to work the best, while the Greedy SPF and Enhanced PUP provide excellent reconstructions as well.

It should be noted that the optimal solution may not necessarily result in reconstruction of the original images. However, if candidate weights have been properly assigned, then the optimal solution should have a large number of fragments in or almost in the right place. Hence, it would be perhaps better for an automated image reassembly tool to present to the forensic analyst a small number of most likely reorderings, based on which the correct reordering can be manually arrived at.

Finally, while we used 24-bit Windows BMPs for the experiments, many other image formats could have been used. In the case of JPEG's additional steps of

decompression, denormalization and inverse DCT of each block would be required. Our algorithms should work with almost any image format that creates a boundary with the next fragment in the image sequence. So while additional steps and modifications must be made for other image formats, any image format that allows us to use boundaries to compare two fragments is potentially reconstructable by our methods. In its current form most progressive image formats like JPEG 2000 will not work with our methods as is. We propose to do additional research with these image formats in future work.

Table 6.2: Image Dataset Reassembly Information

Dataset/ Algorithm	<i>Reconstructed Iteration</i>		<i>Total</i>	<i>Total</i>
	<i>1st</i>	<i>2nd</i>	<i>Iterations</i>	<i>Reconstructed</i>
Caps On The Wall				
Greedy NUP	0	0	1	0
Enh. Greedy NUP	1	0	1	1
Greedy PUP	0	0	1	0
Enh. PUP	1	0	1	1
Greedy SPF	0	0	1	0
Enh. Greedy SPF	1	0	1	1
USC-SIPI				
Greedy NUP	3	1	2	4
Enh. Greedy NUP	3	1	2	4
Greedy PUP	4	0	1	4
Enh. PUP	4	0	1	4
Greedy SPF	2	0	2	2
Enh. Greedy SPF	2	0	2	2
Cricketers				
Greedy NUP	21	1	3	24
Enh. Greedy NUP	21	0	2	21
Greedy PUP	21	0	1	21
Enh. PUP	21	0	1	21
Greedy SPF	22	2	2	24
Enh. Greedy SPF	22	2	2	24
FBI				
Greedy NUP	7	0	2	7
Enh. Greedy NUP	7	0	2	7
Greedy PUP	7	0	2	7
Enh. PUP	8	0	2	8
Greedy SPF	7	0	2	7
Enh. Greedy SPF	10	0	1	10
Aircraft				
Greedy NUP	4	1	2	5
Enh. Greedy NUP	5	0	1	5
Greedy PUP	5	0	1	5
Enh. PUP	5	0	1	5
Greedy SPF	5	0	1	5
Enh. Greedy SPF	5	0	1	5

Table 6.3: Image Dataset Reassembly Information Continued

Dataset/ Algorithm	<i>Reconstructed Iteration</i>		<i>Total Iterations</i>	<i>Total Reconstructed</i>
	<i>1st</i>	<i>2nd</i>		
Nature				
Greedy NUP	1	0	2	1
Enh. Greedy NUP	0	0	1	0
Greedy PUP	1	0	2	1
Enh. PUP	0	0	1	0
Greedy SPF	0	0	1	0
Enh. Greedy SPF	0	0	1	0
Mixed				
Greedy NUP	9	1	2	10
Enh. Greedy NUP	10	0	1	10
Greedy PUP	10	0	1	10
Enh. PUP	10	0	1	10
Greedy SPF	10	0	1	10
Enh. Greedy SPF	10	0	1	10

Table 6.4: Best Reconstruction Algorithm for Datasets

Dataset	<i>Best Algorithm for reconstruction on first attempt</i>	<i>Reconstructed on first attempt</i>
Caps	All algorithms using Enhanced Greedy	1 of 1
USC-SIPI	Greedy PUP & Enhanced PUP	4 of 4
Cricketers	Greedy SPF & Enhanced Greedy SPF	22 of 24
FBI	Enhanced Greedy SPF	10 of 10
Aircraft	All but Greedy NUP	5 of 5
Nature	Greedy and PUP	1 of 5
Mixed	All but Greedy NUP	10 of 10

Table 6.5: Algorithm Percentage Reconstructions

Algorithm	<i>Total Reconstructed</i>	<i>Percentage Reconstructed</i>
Greedy NUP	51	86.4%
Enhanced Greedy NUP	48	81.4%
Greedy PUP	49	81.4%
Enhanced Greedy PUP	49	83.0%
Greedy SPF	48	81.4%
Enhanced Greedy SPF	52	88.1%

Chapter 7

Conclusion

We have introduced and discussed a general procedure for automated re-assembly of scattered image evidence. Experimental results show that even by using a simple greedy algorithm where the best candidate probabilities are used results in most images being reconstructed in their entirety. However by making the enhancements to the greedy algorithm and then using simultaneous reassembly techniques or SPF algorithms we can further improve the reassembly results.

Even those few images that are not reconstructed in their entirety tend to have a large number of fragments that are in the correct order. This is helpful because, if an analyst can identify proper subsequences in these candidate reorderings, they can combine these subsequences to form unit fragments and iterate the process to eventually converge on the proper reordering with much less effort than if they were to perform the task manually.

In future work we will extend the techniques presented in this thesis to reconstruction of shredded documents. We shall also investigate methods to collate fragments of documents from mixed fragments of several documents.

Bibliography

- [1] Department Of Justice. Searching and Seizing Computers and Obtaining Evidence in Criminal Investigations. DOJ [Online]. Available: www.usdoj.gov/criminal/cybercrime
- [2] R. Sablatnig and C. Menard. "On Finding Archaeological Fragment Assemblies Using a Bottom-Up Design," in *Proc. of the 21st Workshop of the Austrian Association for Pattern Recognition Hallstatt*, Austria, Oldenburg, Wien, Muenchen, 1997, pp. 203–207.
- [3] M. Kampel, R. Sablatnig, and E. Costa. "Classification of archaeological fragments using profile primitives," in *Computer Vision, Computer Graphics and Photogrammetry — a Common Viewpoint, Proceedings of the 25th Workshop of the Austrian Association for Pattern Recognition (OAGM)*, 2001, pp. 151–158.
- [4] W. P. Stemmer. "DNA shuffling by random fragmentation and reassembly: in vitro recombination for molecular evolution," in *Proc Natl Acad Sci U S A.*, October 25, 1994.
- [5] H. C. da Gama Leito, and J. Stolfi. "A multiscale method for the reassembly of two-dimensional fragmented objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, September 2002.
- [6] H. C. da Gama Leito and J. Stolfi. "A multi-scale method for the re-assembly of fragmented objects," in *Proc. British Machine Vision Conference - BMVC 2000*, 2000, pp. 705–714.
- [7] H. C. da Gama Leito and J. Stolfi. "Automatic reassembly of irregular fragments," Univ. of Campinas, Tech. Rep. IC-98-06, 1998.

- [8] T. Altman. "Solving the jigsaw puzzle problem in linear time," *Applied Artificial Intelligence*, v.3 n.4, pp. 453–462, 1989.
- [9] D. A. Kosiba, P. M. Devaux, S. Balasubramanian, T. Gandhi and R. Kasturi. "An Automatic Jigsaw Puzzle Solver," in *Proc. Int. Conf. Pattern Recognition*, Jerusalem, 1994, pp. 616–618.
- [10] G. C. Burdea and H. J. Wolfson. "Solving jigsaw puzzles by a robot," *IEEE Transactions on Robotics and Automation*, pp. 5(6):752764, 1989.
- [11] M. G. Chung, M. Fleck and D. A. Forsyth. "Jigsaw Puzzle Solver Using Shape and Color," in *Proc. ICSP-T98*, 1998, pp. 877–880.
- [12] C. E. Leiserson et. al. "Introduction to algorithms," *MIT Press*, 2001.
- [13] S. A. Martucci. "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," in *IEEE International Symposium on Circuits and Systems*, 1990, pages 1310–1313.
- [14] O. de Vel. "File classification using byte sub-stream kernels," *Journal of Digital Investigation*, Vol. 1, No. 2, 2004.
- [15] F. Amigoni, S. Gazzani, S. Podico. "A Method for Reassembling Fragments in Image Reconstruction," Presented at the *International Conference on Image Processing*, Barcelona, Spain, 2003.
- [16] K. Shanmugasundaram and N. Memon. "Automatic Reassembly of Document Fragments via Data Compression," Presented at the *2nd Digital Forensics Research Workshop*, Syracuse, July 2002.
- [17] A. Pal, K. Shanmugasundaram and N. Memon. "Automated Reassembly of Fragmented Images," Presented at *ICASSP*, 2003.
- [18] J. Vygen. "Disjoint paths," Research Institute for Discrete Mathematics, University of Bonn, Tech. Rep. 94816, 1994.

- [19] E. W. Dijkstra. "A note on two problems in connexion with graphs," *Numerische Mathematik*, pp. 1:269–271, 1959.
- [20] S. G. Koliopoulos and C. Stein. "Approximating disjoint-path problems using greedy algorithms and packing integer programs," in *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference IPCO VI, Lecture Notes in Computer Science 1412*, 1998, pp. 152–168.
- [21] J. M. Kleinberg, "Approximation algorithms for disjoint paths problems," Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Ph.D Thesis, 1996.
- [22] P. Carmi, T. Erlebach and Y. Okamoto. "Greedy edge-disjoint paths in complete graphs," Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Tech. Rep: 155, February 2003.
- [23] N. Robertson and P. D. Seymour. "Outline of a disjoint paths algorithm," in B. Korte, L. Lovasz, H. J. Promel, and A. Schrijver, eds., *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.
- [24] A. Schrijver. "Homotopic routing methods," In B. Korte, L. Lovasz, H. J. Promel, and A. Schrijver, eds., *Paths, Flows and VLSI-Layout*. Springer, Berlin, 1990.