Automated Reassembly of File Fragmented Images Using Greedy Algorithms

Nasir Memon, *Member, IEEE,* and Anandabrata Pal

***Abstract*—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. In this paper, we investigate 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 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 provide techniques for comparing fragments and describe several algorithms for image reconstruction based on greedy heuristics. Finally, we provide experimental results showing that images can be reconstructed with high accuracy even when there are thousands of fragments and multiple images involved.**

***Index Terms*—File fragmentation, forensics, greedy algorithms, reassembly.**

I. Introduction

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. 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 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]–[10]. The digital forensic equivalent of the reconstruction of fragmented objects problem, which we call *reassembling fragmented documents*, however, has received little attention.

In this paper, 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 nonimage fragments from the set of fragments to analyze.

The remainder of this paper is structured as follows. In the next section, we briefly explain how fragmentation can occur. Then in Section III we formulate the problem as a combinatorial optimization problem and present general techniques for image reassembly. Section IV presents experimental results and we conclude in Section V with a discussion on future work.

II. Scenarios Causing Fragmentation

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. 1 is a very simple example of a disk with ten clusters. Fig. 1(a) displays four image files A, B, C, and D which are

Manuscript received September 20, 2004; revised March 17, 2005. This work was supported in part by AFOSR under Grant F49620-01-1-0243. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Gaurav Sharma.

The authors are with the Computer Science Department, Polytechnic University, Brooklyn, NY 11201 USA (e-mail: memon@poly.edu; pashapal@aol.com).

Digital Object Identifier 10.1109/TIP.2005.863054

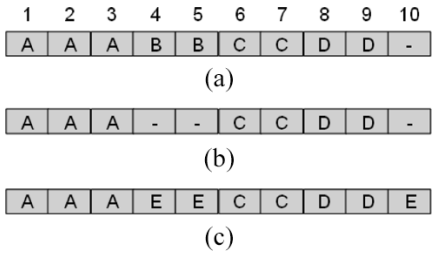


Fig. 1. Simplified example of file fragmentation. (a) Unfragmented disk with ten clusters (cluster 10 has no data). (b) Free space fragmentation after file B is deleted. (c) Fragmentation of file E.

stored in consecutive clusters. This means there is no fragmentation. In Fig. 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 three clusters to store. The OS then saves the file in clusters 4, 5, and 10, as shown in Fig. 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.

When deleting files, an OS 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.

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 10-GB 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 section introduces the reassembly problem formally, and describes our approaches to solve the problem.

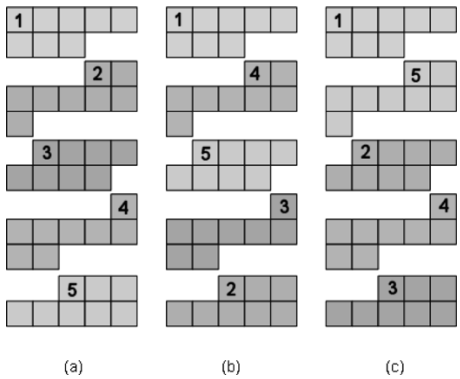


Fig. 2. (a)–(c) Three different possibilities for the reconstruction of an image with five fragments.

III. Reassembly Problem

In this section, we formulate the image reassembly problem in a more rigorous manner and describe several approaches for a solution to the problem.

A. Statement of the Problem

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. 2 shows three potential reassembly sequences of an image consisting of five 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.

We begin with evaluating the case of a single image split into multiple fragments. Suppose we have a set $\{A_0, A_1 \ldots, A_n\}$ of fragments of an image A. We would like to compute a permutation π such that $A = A_{\pi(0)}||A_{\pi(1)}|| \ldots ||A_{\pi(n)}$, 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.

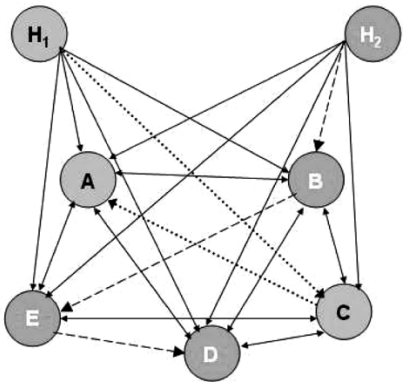


Fig. 3. Graph of seven fragments and two vertex disjoint paths (H_1CA and H_2BED).

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

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

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 (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].

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.

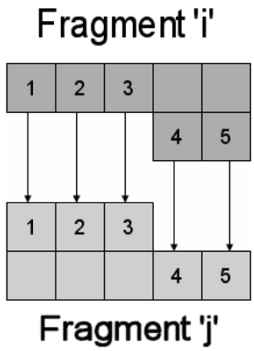


Fig. 4. Pixel values being compared when calculating candidate weights between two fragments.

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 ith image, then the correct solutions $(P_1, P_2 \ldots, P_k)$ 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) 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].

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 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.

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 section.

B. Assigning Adjacency Weights

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). 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 three 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 the width (w) is 5 and PM would compare to see if each numbered pixel 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 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. Now that we can calculate candidate weights between each pair of fragments, we need algorithms to reconstruct the images based on these weights.

C. Reconstruction Algorithms

We are now ready to present algorithms to reconstruct the images based on the candidate weights between fragments. This section describes eight 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 [23], [24]. The use of greedy approximation algorithms to solve edge and vertex disjoint problems has been studied extensively [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 nonunique 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.

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.

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 $O(nlogn)$ 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 $O(nlogn)$ steps. However, we have to sort for every fragment so the complexity will be $O(n^2logn)$ for the sorting step. We now present four 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, 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. 5, with greedy SUP, both the image of the dog and plane will reconstruct perfectly if the dog is reconstructed first and then the plane (5a). If the plane is reconstructed first it will reconstruct incorrectly thus causing the dog to reconstruct incorrectly (5b). 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 $O(n) + O(n^2) + O(n^2logn) = O(n^2logn)$.

2) **Greedy NUP**: Greedy nonunique 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.

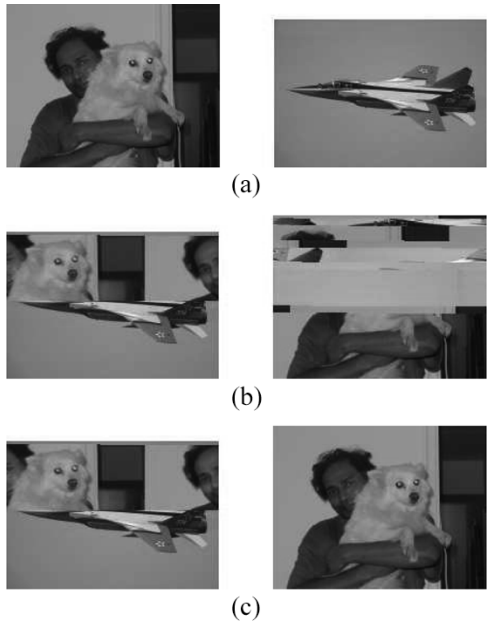


Fig. 5. Example of reassembly using UP and NUP which can be rectified with the enhanced greedy based algorithms. Images from the mixed dataset. (a) Proper reassembly of images with greedy UP when processing dog first. (b) Improper reassembly of images with greedy UP when processing plane first. (c) Improperly reassembled image of plane with greedy NUP.

   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. 5(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 NUP's complexity is the same as greedy SUP $O(n^2logn)$.

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 $S = (s_1, s_2, \ldots, s_k)$

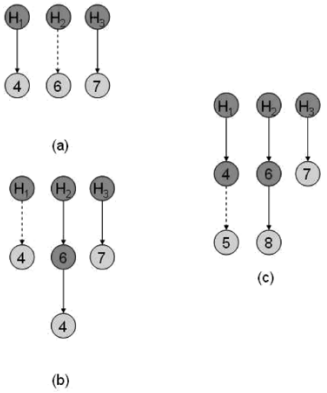


Fig. 6. Parallel unique path (PUP) algorithm example.

of current fragments for processing where s_i is the current fragment for the ith 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 $T = (t_1, t_2, \ldots, t_k)$ 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 ith image ($P_i = P_i || t_i$), replace the current fragment for the ith 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. 6 shows an example of the algorithm where there are three images being reconstructed. Fig. 6(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. 6(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 (6c). This process continues until all images are reconstructed.

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 $O(n^2 logn)$.

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 $O(n^2 logn)$.

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.

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 $O(n^2 logn)$ 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.

IV. 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 three iterations were needed to reconstruct the majority of images in almost all cases. Table I shows the general image statistics for the seven datasets, and Table II shows the number of images reconstructed using each algorithm. Table III shows the best algorithms for reassembling

TABLE I
General Dataset Image Information

Dataset	*Images*	*Resolution*	*Fragments*
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

TABLE II
Image Dataset Reassembly Information

Dataset/ Algorithm	*Reconstructed Iteration* 1^{st}	2^{nd}	*Total Iterations*	*Total Reconstructed*
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
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 III
BEST RECONSTRUCTION ALGORITHM FOR DATASETS

Dataset	*Best Algorithm for reconstruction on first attempt*	*Reconstructed on first attempt*
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 IV
ALGORITHM PERCENTAGE RECONSTRUCTIONS

Algorithm	*Total Reconstructed*	*Percentage Reconstructed*
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%

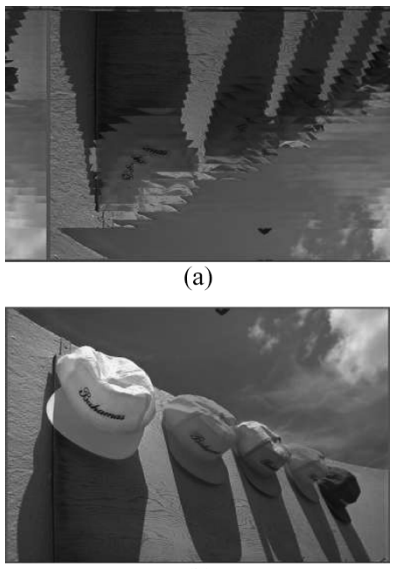


Fig. 7. Example of proper and improper reassemblies of the Caps image. (a) Caps image with fragments out of order and (b) same image with fragments in correct order.

each dataset on the first iteration. Finally, Table IV 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 768 $\times$ 512 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. 7).

The second data set used was a collection of 4 512 $\times$ 512 images from the USC-SIPI Image Database. The images used were

Fig. 8. Reassembly of FBI most wanted using different algorithms.

Girl (Lena), Baboon, Airplane (F-16), and Sailboat on lake. The two PUP algorithms resulted in perfect reconstruction of all four 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 two or three images not reconstructed perfectly look visually to be perfect, the last fragments containing hair standing up were swapped.

The fourth data set that was used consisted of ten images retrieved from the FBI's most wanted Web site (Fig. 8). The various algorithms worked with different levels of accuracy. The best algorithm was the enhanced greedy SPF algorithm which resulted in all ten images being reconstructed.

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

The sixth data set used was a collection of five nature scene images. Only one 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.

The final set used was a collection of ten relatively nonrelated images. There were four images of fighter planes mixed in with one image of an actress posing, one 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 ten 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 JPEGs 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.

V. Conclusion

We have introduced and discussed a general procedure for automated reassembly 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 paper to reconstruction of shredded documents. We shall also investigate methods to collate fragments of documents from mixed fragments of several documents.

References

[1] U.S. Department Of Justice, "Searching and Seizing Computers and Obtaining Evidence in Criminal Investigations," [Online] Available, http://www.usdoj.gov/criminal/cybercrime.

[2] R. Sablatnig and C. Menard, "On finding archaeological fragment assemblies using a bottom-up design," in *Proc. 21st Workshop Austrian Association for Pattern Recognition Hallstatt*, Oldenburg, Austria, 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, Proc. 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. Nat. Acad. Sci.*, Oct. 25, 1994.

[5] H. C. da Gama Leito and J. Soltfi, "A multiscale method for the reassembly of two-dimensional fragmented objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, pp. 1239–1251, Sep. 2002.

[6] H. C. da Gama Leito and J. Stolfi, "A multi-scale method for the re-assembly of fragmented objects," in *Proc. Br. Machine Vision Conf. 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," *Appl. Artif. Intell.*, vol. 3, no. 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, Israel, 1994, pp. 616–618.

[10] G. C. Burdea and H. J. Wolfson, "Solving jigsaw puzzles by a robot," *IEEE Trans. Robot. Automat.*, vol. 5, no. 6, pp. 752–764, 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*. Cambridge, MA: MIT Press, 2001.

[13] S. A. Martucci, "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1990, pp. 1310–1313.

[14] O. de Vel, "File classification using byte sub-stream kernels," *J. Dig. Investigation*, vol. 1, no. 2, 2004.

[15] F. Amigoni, S. Gazzani, and S. Podico, "A method for reassembling fragments in image reconstruction," in *Proc. Int. Conf. Image Processing*, Barcelona, Spain, 2003.

[16] K. Shanmugasundaram and N. Memon, "Automatic reassembly of document fragments via data compression," in *Proc. 2nd Digital Forensics Research Workshop*, Syracuse, NY, Jul. 2002.

[17] A. Pal, K. Shanmugasundaram, and N. Memon, "Automated reassembly of fragmented images," in *Proc. ICASSP*, 2003.

[18] J. Vygen, "Disjoint paths," Res. Inst. Discrete Mathemathics, Univ. Bonn, Bonn, Germany, Tech. Rep. 94 816, 1994.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, pp. 1:269–1:271, 1959.

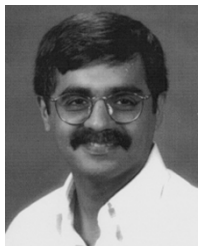
[20] S. G. Koliopoulos and C. Stein, "Approximating disjoint-path problems using greedy algorithms and packing integer programs," in *Proc. 6th Integer Programming and Combinatorial Optimization Conf. VI, Lecture Notes in Computer Science*, vol. 1412, 1998, pp. 152–168.

[21] J. M. Kleinberg, "Approximation algorithms for disjoint paths problems," Ph.D dissertation, Dept. Elect. Eng. Comput. Sci., Mass. Inst. Technol., Cambridge, 1996.

[22] P. Carmi, T. Erlebach, and Y. Okamoto, "Greedy edge-disjoint paths in complete graphs," Computer Engineering and Networks Laboratory (TIK), Zurich, Switzerland, Tech. Rep. 155, Feb. 2003.

[23] N. Robertson and P. D. Seymour, "Outline of a disjoint paths algorithm," in *Paths, Flows and VLSI-Layout*, B. Korte, L. Lovasz, H. J. Promel, and A. Schrijver, Eds. Berlin, Germany: Springer-Verlag, 1990.

[24] A. Schrijver, "Homotopic routing methods," in *Paths, Flows and VLSI-Layout*, B. Korte, L. Lovasz, H. J. Promel, and A. Schrijver, Eds. Berlin, Germany: Springer, 1990.



Nasir Memon (S'91–M'92) is a Professor in the Computer Science Department, Polytechnic University, New York. His research interests include data compression, computer and network security, and multimedia communication, computing, and security. He has published more than 150 articles in journals and conference proceedings. He was a visiting faculty at Hewlett-Packard Research Labs during the academic year 1997–1998. He is currently an associate editor for *ACM Multimedia Systems Journal* and the *Journal of Electronic Imaging*.

Dr. Memon has won several awards including the NSF CAREER award and the Jacobs Excellence in Education award. He was an Associate Editor for the IEEE Transactions on Image Processing from 1999 to 2002.

Anandabrata Pal received the B.S. degree in computer science from the New York Institute of Technology in 2000.

He is currently a Research Fellow at Polytechnic University, New York. He has conducted research and published papers in image file fragmentation, obfuscation, and video. His work has also been extended into the preliminary research for reassembly of shredded documents.