A Novel Inequality-Based Fragmented File Carving Technique

Hwei-Ming Ying and Vrizlynn L.L. Thing

Institute for Infocomm Research, Singapore
{hmying,vriz}@i2r.a-star.edu.sg

Abstract. Fragmented File carving is an important technique in Digital Forensics to recover files from their fragments in the absence of the file system allocation information. In this paper, the fragmented file carving problem is formulated as a graph theoretic problem. Using this model, we describe two algorithms, "Best Path Search" and "High Fragmentation Path Search", to perform file reconstruction and recovery. The best path search algorithm is a deterministic technique to recover the best file construction path. We show that this technique is more efficient and accurate than existing brute force techniques. In addition, a test was carried out to recover 10 files scattered into their fragments. The best path search algorithm was able to successful recover all of them back to their original state. The high fragmentation path search technique involves a trade-off between the final score of the constructed path of the file and the file recovery time to allow a faster recovery process for highly fragmented files. Analysis show that the accurate eliminations of paths have an accuracy of up to greater than 85%.

1 Introduction

The increasing reliance on digital storage devices such as hard disks and solid state disks for storing important private data and highly confidential information has resulted in a greater need for efficient and accurate data recovery of deleted files during digital forensic investigation.

File carving is the technique to recover such deleted files, in the absence of file system allocation information. However, there are often instances where files are fragmented due to low disk space, file deletion and modification. In a recent study [10], FAT was found to be the most popular file system, representing 79.6% of the file systems analyzed. From the files tested on the FAT disks, 96.5% of them had between 2 to 20 fragments. This scenario of fragmented and subsequently deleted files presents a further challenge requiring a more advanced form of file carving techniques to reconstruct the files from the extracted data fragments.

The reconstruction of objects from a collection of randomly mixed fragments is a common problem that arises in several areas, such as archaeology [9], [12], biology [15] and art restoration [3], [2]. In the area of fragmented file craving, research efforts are currently on-going. A proposed approach is known as the Bifragment gap carving(BGC) [13]. This technique searches and recovers files, fragmented into two fragments that contain identifiable headers and footers. An idea of using a graph theoretic approach to perform file craving has also been studied in [8], [14], [4] and [5]. In graph theoretic carving, the fragments are represented by the vertices of a graph and the edges are assigned weights which are values that indicate the likelihood that two fragments are adjacent in the original file. For example in image files, we list two possible techniques to evaluate the candidate weighs between any two fragments [8]. The first is pixel matching whereby the total number of pixels matching along the edges for the two fragments are summed. Each pixel value is then compared with the corresponding pixel value in the other fragment. The closer the values, the better the match. The second is median edge detection. Each pixel is predicted from the value of the pixel above, to the left and left diagonal to it [11]. Using median edge detection, we would sum the absolute value of the difference between the predicted value in the adjoining fragment and the actual value. The carving is then based on obtaining the path of the graph with the best set of weights. In addition, Cohen, 2007 introduced a technique of carving involving mapping functions and discriminators in [6], [7]. These mapping functions represent various ways for which a file can be reconstructed and the discriminators will then check on the validity of them until the best one is obtained. We discuss these methods further in Section 3 on related work.

In this paper, we model the problem in a graph theoretic form which is not restricted by the limitation of the number of fragments. We assume that all the fragments belonging to a file are known. This can be achieved through identification of fragments for a file based on groups of fragments belonging to an image of same scenery (i.e. edge pixel difference detection) or context based modelling for document fragments [4].

We define a file construction path as one passing through all the vertices in the graph. In a graph, there are many different possible file construction paths. An optimal path is one which gives the largest sum of weight (i.e. final score) for all the edges it passes through. The problem of finding the optimum path is intractable [1]. Furthermore, it is well known that applying the greedy algorithm does not give good results and that computing all the possible paths is resource-intensive and not feasible for highly fragmented files. In this paper, we present two main algorithms namely the "Best Path Search" and the "High Fragmentation Path Search". Best Path search is an inequality-based method which will reduce the required computations. This algorithm is more efficient and faster than brute force which computes all the possible path combinations. It is suitable for relative small values of n. For larger values of n, we introduce the High Fragmentation Path Search, which is a tradeoff algorithm to allow a flexible control over the complexity of the algorithm, while at the same time, obtain sufficiently good results for fragmented file carving.

2 Statement of Problem

In fragmented file carving, the objective is to arrange a file back to its original structure and recover the file in as short a time as possible. The technique should not rely on the file system information, which may not exist (e.g. deleted fragmented file, corrupted file system). We are presented with files that are not arranged in its proper original sequence from its fragments. The goal in this paper is to arrange them back to its original state in a short a time as possible.

The core approach would be to test each fragment against one another to check how likely any two fragments is a joint match. They are then assigned weights and these weights represent the likelihood that two fragments are a joint match. Since the header can be easily identified, any edge joining the header is considered a single directional edge while all other edges are bi-directional. Therefore, if there are n fragments, there will be a total of $(n\text{-}1)^2$ weights. The problem can thus be converted into a graph theoretic problem where the fragments are represented by the vertices and the weights are represented by the edges. The goal is to find a file construction path which passes each vertex exactly once and has a maximum sum of edge weights, given the starting vertex. In this case, the starting vertex will correspond to the header.

A simple but tedious approach to solve this problem is to try all path combinations, compute their sums and obtain the largest value which will correspond to the path of maximum weight. Unfortunately, this method will not scale well when n is large since the number of computations of the sums required will be $(n\text{-}1)!$. This complexity increases exponentially as n increases.

3 Related Work

Bifragment gap carving [13] was introduced as a fragmented file carving technique that assumed most fragmented files comprise of the header and footer fragments only. It exhaustively searched for all the combinations of blocks between an identified header and footer, while incrementally excluded blocks that result in unsuccessful decoding/validation of the file. A limitation of this method was that it could only support carving for files with two fragments. For files with more than two fragments, the complexity could grow extremely large.

Graph theoretic carving was implemented as a technique to reassemble fragmented files by constructing a k-vertex disjoint graph. Utilizing a matching metric, the reassembly was performed by finding an optimal ordering of the file blocks/sectors. The different graph theoretic file carving methods are described in [8]. The main drawback of the greedy heuristic algorithms was that it failed to obtain the optimal path most of the time. This was because they do not operate exhaustively on all the data. They made commitments to certain choices too early which prevented them from finding the best path later.

In [6], the file fragments were "mapped" into a file by utilizing different mapping functions. A Mapping function generator generated new mapping functions which were tested by a discriminator. The goal of this technique was to derive a mapping function which minimizes the error rate in the discriminator. It is of great importance to construct a good discriminator for it to localize errors within the file, so that discontinuities can be determined more accurately. If the discriminator failed to indicate the precise locations of the errors, then all the permutations need to be generated which could become intractable.

4 Inequality-Based File Carving Technique

The objective of our work is to devise a method to produce the optimum file construction path and yet achieve a lesser complexity than the brute force approach which requires the computation of all possible paths.

In this section, we do an investigation of the non-optimal paths that can be eliminated. In doing so, the complexity can be reduced when doing the final evaluations of possible candidates for the optimal path. The general idea is described below.

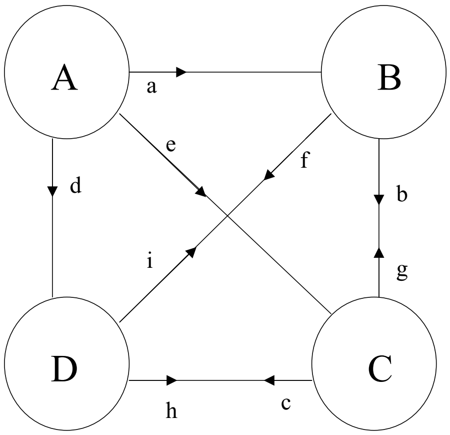

Fig. 1. n=4 (General Case)

In Figure 1, we show an example of a file with 4 fragments (n=4). A, B, C and D represent the file fragments. The letters, a to i, assigned to the edges represent the numbered values of the likelihood of a match between two adjacent fragments in a particular direction. Assume that A is the header fragment which can be easily identified. Let *f(x)* represent the sum of the edges of a path where x is a path. Computing the values of *f(x)* for all the possible paths, we obtain:

f(ABCD) = $a + b + c$
f(ABDC) = $a + f + h$
f(ACBD) = $e + g + f$
f(ACDB) = $e + c + i$
f(ADBC) = $d + i + b$
f(ADCB) = $d + h + g$

Arrange the values of each individual a to i in ascending order. From this chain of inequalities formed from these nine variables, it is extremely unlikely that the optimal path can identified immediately except in very rare scenarios. However, it is possible to eliminate those paths (without doing any additional computations) which we can be certain are non optimal. The idea is to extract more information that can be deduced from the construction of these inequalities. Doing these eliminations will reduce the number of evaluations which we need to compute at the end and hence will result in a reduction in complexity while still being able to obtain the optimal path.

5 Best Path Search Algorithm

The general algorithm is as follows:

1) For a fixed n, assign $(n\text{-}1)^2$ variables to the directed edges.
2) Work out f(each path) in terms of the sum of n-1 of these variables and arrange the summation in ascending order.
3) Establish the chain of inequalities based on the actual values of the directed edges.
4) Pick the smallest value and identify the paths which contain that value.
5) Do a comparison of that path with other paths at every position of the summation. If the value at each position of this path is less than the corresponding positions with any other path, then the weaker path that has been chosen can be eliminated.
6) Repeat steps 4 to 6 for other paths to determine if they can be eliminated.
7) The remaining paths that remain are then computed to determine the optimal path.

6 Analysis of Best Path Search Algorithm

The algorithm is an improvement over the brute force method in terms of reduced complexity and yet can achieve a 100% success rate of obtaining the optimal path.

Let n = 3. Assign four variables, a, b, c, d to the four directed weights. There are a total of 4! = 24 ways in which the chain of inequality can be formed. Without loss of generality, we can assume that the values of the 2 paths are a+c and b+d. Hence, there are a total of 8 possible chains of inequalities such that no paths can be eliminated. This translates to a probability of $\frac{8}{24} = \frac{1}{3}$. Therefore, there is a probability of $\frac{1}{3}$ that 2 computations are necessary to evaluate the optimal paths and a probability of $\frac{2}{3}$ that no computations are needed to do likewise. Hence, the average complexity required for the case n = 3 is $\frac{1}{3} * 2 + \frac{2}{3} * 0 = \frac{2}{3}$. Since brute force requires 2 computations, this method of carving on average will require only 33% of the complexity of brute force.

To calculate an upper bound for the number of comparisons needed, assume that every single variable of all possible paths have to compared against one another. Since there are (n-1)! possible paths and each path contains (n-1) variables, an upper bound for the number of comparisons required

$= (\text{n-1})!* \frac{[(n-1)!-1]}{2} * (\text{n-1})$
$= (\text{n-1})!* (\text{n-1})* \frac{[(n-1)!-1]}{2}$

For general n, when all the paths are written down in terms of their variables, it is observed that each path has exactly n -1 other paths such that they have one variable in common.

By using the above key observation, it is possible to evaluate the number of pairs of paths such that they have a variable in common.

No. of pairs of paths such that they have a variable in common = (n-1)! * $\frac{n-1}{2}$
Since there are a total of (n-1)!* $\frac{(n-1)!-1}{2}$ possible pair of paths, the percentage of pairs of paths which will have a variable in common = $\frac{100n-100}{(n-1)!-1}\%$

The upper bound which was obtained earlier can now be strengthened to

(n-1)!* (n-1)* $\frac{(n-1)!-1}{2}$ - (n-1)! * $\frac{n-1}{2}$
= (n-1)!* (n-1)* $\frac{(n-1)!-2}{2}$

The implementation to do these eliminations is similar to the general algorithm given earlier but with the added step of ignoring the extra comparison whenever a common variable is present. For any general n, apply the algorithm to determine the number of paths k that cannot be eliminated. This value of k will depend on the configurations of the weights given.

To compute the time complexity of this carving method, introduce functions g(x) and h(x) such that g represents the time taken to do x comparisons and h represents the time taken to do x summations of (n-1) values.

The least number of comparisons needed such that k paths remain after implementing the algorithm

= [(n-1)! - k]* (n-1) + $\frac{k(k-1)}{2}$
= (n-1)!* (n-1) - k* (n-1) + $\frac{k(k-1)}{2}$
= (n-1)!* (n-1) + k* (k-3)* $\frac{n-1}{2}$
= (n-1)[(n-1)! + $\frac{k(k-3)}{2}$]

The greatest number of comparisons needed such that k paths remain after implementing the algorithm

= [(k-1) * (n-1)! - $\frac{k(k-1)}{2}$]* (n-1) + [(n-1)! - k]* (n-1)
= (n-1)[k*(n-1)! - $\frac{k(k-1)}{2}$]

Hence, the average number of comparisons needed in the implementation

= 1/2 * (n-1)[(n-1)! + $\frac{k(k-3)}{2}$] + 1/2 * (n-1)[k*(n-1)! - $\frac{k(k-1)}{2}$]
= (n-1)* [(k+1)* $\frac{(n-1)!}{2}$ - k]

The total average time taken to implement the algorithm is equal to the sum of the time taken to do the comparisons and the time taken to evaluate the remaining paths

= g((n-1)* [(k+1)* $\frac{(n-1)!}{2}$ - k]) + h(k)

Doing comparisons of values take a shorter time compared to evaluating the sum of n-1 values and hence, the function g is much smaller than the function h. Thus, this time complexity can be approximated to be h(k) and since h(k) < h((n-1)!), this carving method is considerably better than brute force.

A drawback of this method is that even after the eliminations, the number of paths that need to be computed might still be exceedingly large. In this case, we can introduce a high fragmentation path search algorithm as described below.

7 High Fragmentation Path Search Algorithm

In the previous sections, we introduced a deterministic way of obtaining the best path. It is suitable for relatively small values of n where the computational complexity is minimal. For larger values of n, we propose a probabilistic algorithm which offers a tradeoff between obtaining the best path and the computational complexity.

The algorithm is described as follows.

1) For a fixed n, assign $(n-1)^2$ variables to the directed edges.
2) Work out f(each path) in terms of the sum of n-1 of these variables and arrange the summation in ascending order.
3) Establish the chain of inequalities based on the actual values of the directed edges.
4) Pick the smallest value and identify the paths which contain that value.
5) Do a comparison of that path with other paths at every position of the summation. If the value at each position of this path is less than the corresponding positions with any other path, then the weaker path that has been chosen can be eliminated.
6) Repeat steps 4 to 6 for other paths to determine if they can be eliminated.
7) The remaining paths are then compared pairwise at their corresponding positions. The ones that have lesser values in more positions are then eliminated.
8) If both the paths have an equal number of lesser and greater values at the corresponding positions, then neither of the paths are eliminated.
9) Repeat step 7 for the available paths until the remaining number of paths is a small enough number to do computations.
10) Compute all remaining paths to determine "optimal path"

This probabilistic algorithm is similar to the general algorithm from step 1 to 6. The additional steps 7 to 9 are added to reduce the complexity of the algorithm.

8 Analysis of High Fragmentation Path Algorithm

We shall use a mathematical statistical method to do the analysis of the general case. Instead of arranging the variables of each path in ascending order, we can also skip this step which will save a bit of time. So now instead of comparing the variables at each position between 2 paths, we can just take any variable from each path at any position to do the comparison.

Since the value of each variable is uniformly distributed in the interval (0,1), the difference of two such independent variables will result in a triangular distribution. This triangular distribution has probability density function of f(x) = 2 - 2x and a cumulative distribution function of 2x - x^2. Its expected value is $\frac{1}{3}$ and its variance is $\frac{1}{18}$. Let the sum of the edges of a valid path A be $x_1 + x_2 + ... + x_{n-1}$ and let the sum of edges of a valid path B be $y_1 + y_2 + ... + y_{n-1}$ where n is the number of fragments to be recovered including the header. If $x_i - y_i > 0$ for more than $\frac{n-1}{2}$ values of i, then we eliminate path B. Similarly, if path $x_i - y_i < 0$ for less than $\frac{n-1}{2}$ values of i, then we eliminate path A. The aim is to evaluate the probability of f(A) > f(B) in the former case and the probability of f(A) < f(B) in the latter case. Assume $x_i - y_i > 0$ for more than $\frac{n-1}{2}$ values of i, then we can write $P(x_1 + x_2 + ... + x_{n-1} > y_1 + y_2 + ... + y_{n-1}) = P(M > N)$ where M is the sum of all $z_i = x_i - y_i > 0$ and N is the sum of all $w_i = y_i - x_i > 0$. From the assumption, the number of variables in M is greater than the number of variables in N. Both z_i and w_i in both M and N are random variables of triangular distribution and thus since the sum of independent random variable with a triangular distribution approximates to a normal distribution (by the Central Limit Theorem), both Z and W approximates to a normal distribution. Let k be the number of z_i and (n-1-k) be the number of w_i.

Then, the expected value of Z = E(Z) = E(kX) = kE(X) = $\frac{k}{3}$.

The variance of Z = Var(Z) = Var(kX) = k^2Var(X) = $k^2/18$.

Expected values of W = E(W) = E((n-1-k)Y) = (n-1-k)E(Y) = $\frac{n-1-k}{3}$.

Variance of W = Var(W) = Var((n-1-k)Y) = $(n\text{-}1\text{-}k)^2$Var(Y) = $(n\text{-}1\text{-}k)^2/18$.

Hence, the problem of finding $P(x_1 + x_2 + ... + x_{n-1} > y_1 + y_2 + ... + y_{n-1})$ is equivalent to finding the P(Z > W) where Z and W are normally distributed with mean = $\frac{k}{3}$, variance = $k^2/18$ and mean = $\frac{n-1-k}{3}$ and variance = $(n\text{-}1\text{-}k)^2/18$ respectively.

Therefore, P(Z > W) = P(Z - W > 0) = P(U > 0) where U = Z - W. Since U is a difference of two normal distributions, U has a normal distribution with mean = E(Z) - E(W) = $\frac{k}{3}$ - $\frac{n-1-k}{3}$ = $\frac{2k-n+1}{3}$ and

variance = Var(Z) + Var(W) = $k^2/18 + (n\text{-}1\text{-}k)^2/18 = [(n\text{-}1\text{-}k)^2 + k^2]/18$. P(U > 0) can now be found easily since the exact distribution of U is obtained and finding P(W > 0) is equivalent to P(f(A) > f(B)) which gives the probability of f(A) > f(B) (the probability of the value of path A greater than B for a general n).

For example, let n = 20 and k = 15. Then P(f(A) > f(B)) = P(W > 0) where U is normally distributed with mean $\frac{11}{3}$ and variance = $\frac{241}{18}$. Hence, P(W > 0) = 0.8419. This implies that path A has a 84% chance of being the higher valued path compared to path B.

A table for n =30 and various values of k is constructed below:

Table 1. Probability for corresponding k when n=30

k	P(f(A) > f(B))
25	87.96%
24	86.35%
23	84.41%
22	82.09%
21	79.33%
20	76.10%
19	72.33%
18	68.05%

9 Results and Evaluations

We conducted some tests on 10 image files of 5 fragments each. Each pair of directional edge is evaluated and assigned a weight value, with a lower weight representing a higher likelihood of a correct match. The 10 files are named A, B,......, J and the fragments are numbered 1 to 5. X(i,j) denote the edge linking i to j in that order of file X. The original files are in the order of X(1,2,3,4,5) where 1 represents the known header. The results of the evaluation of weights are given in Table 2.

Considering file A, we have the following 24 paths values:

f(12345) = A(1,2) + A(2,3) + A(3,4) + A(4,5)
f(12354) = A(1,2) + A(2,3) + A(3,5) + A(3,4)
f(12435) = A(1,2) + A(2,4) + A(4,3) + A(3,5)
f(12453) = A(1,2) + A(2,4) + A(4,5) + A(5,3)
f(12534) = A(1,2) + A(2,5) + A(5,3) + A(3,4)
f(12543) = A(1,2) + A(2,5) + A(5,4) + A(4,3)
f(13245) = A(1,3) + A(3,2) + A(2,4) + A(4,5)
f(13254) = A(1,3) + A(3,2) + A(2,5) + A(5,4)
f(13425) = A(1,3) + A(3,4) + A(4,2) + A(2,5)
f(13452) = A(1,3) + A(3,4) + A(4,5) + A(5,2)
f(13524) = A(1,3) + A(3,5) + A(5,2) + A(2,4)
f(13542) = A(1,3) + A(3,5) + A(5,4) + A(4,2)
f(14235) = A(1,4) + A(4,2) + A(2,3) + A(3,5)
f(14253) = A(1,4) + A(4,2) + A(2,5) + A(5,3)
f(14325) = A(1,4) + A(4,3) + A(3,2) + A(2,5)
f(14352) = A(1,4) + A(4,3) + A(3,5) + A(5,2)
f(14523) = A(1,4) + A(4,5) + A(5,2) + A(2,3)
f(14532) = A(1,4) + A(4,5) + A(5,3) + A(3,2)
f(15234) = A(1,5) + A(5,2) + A(2,3) + A(3,4)
f(15243) = A(1,5) + A(5,2) + A(2,4) + A(4,3)
f(15324) = A(1,5) + A(5,3) + A(3,2) + A(2,4)
f(15342) = A(1,5) + A(5,3) + A(3,4) + A(4,2)
f(15423) = A(1,5) + A(5,4) + A(4,2) + A(2,3)
f(15432) = A(1,5) + A(5,4) + A(4,3) + A(3,2)

The chain of inequalities is given as below:

A(1,2) < A(2,3) < A(4,5) < A(3,4) < A(1,3) < A(5,3) < A(5,2) < A(3,5) < A(4,2) < A(1,5) < A(4,3) < A(2,5) < A(5,4) < A(1,4) < A(3,2) < A(2,4)

Applying the best path search algorithm will indicate that f(12345) will result in the minimum value among all the paths. Hence, the algorithm outputs the optimal path as 12345 which is indeed the original file. The other files from B to J are done in a similar way and the algorithm is able to recover all of them accurately.

Table 2. Weight values of edges

Edges	Weights	Edges	Weights	Edges	Weights	Edges	Weights	Edges	Weights
A(1,2)	25372	B(1,2)	26846	C(1,2)	1792	D(1,2)	1731	E(1,2)	20295
A(1,3)	106888	B(1,3)	255103	C(1,3)	189486	D(1,3)	169056	E(1,3)	170011
A(1,4)	411690	B(1,4)	238336	C(1,4)	234623	D(1,4)	170560	E(1,4)	461661
A(1,5)	324065	B(1,5)	274723	C(1,5)	130208	D(1,5)	34583	E(1,5)	516498
A(2,3)	27405	B(2,3)	26418	C(2,3)	29592	D(2,3)	11546	E(2,3)	15888
A(2,4)	463339	B(2,4)	211579	C(2,4)	282775	D(2,4)	169162	E(2,4)	404686
A(2,5)	361142	B(2,5)	262210	C(2,5)	259358	D(2,5)	179053	E(2,5)	391823
A(3,2)	421035	B(3,2)	242422	C(3,2)	234205	D(3,2)	168032	E(3,2)	470644
A(3,4)	66379	B(3,4)	37416	C(3,4)	35104	D(3,4)	25275	E(3,4)	33488
A(3,5)	294658	B(3,5)	309995	C(3,5)	278213	D(3,5)	169954	E(3,5)	191333
A(4,2)	322198	B(4,2)	278721	C(4,2)	130525	D(4,2)	34434	E(4,2)	521456
A(4,3)	358088	B(4,3)	259830	C(4,3)	261451	D(4,3)	176501	E(4,3)	395452
A(4,5)	57753	B(4,5)	19728	C(4,5)	20939	D(4,5)	1484	E(4,5)	12951
A(5,2)	279017	B(5,2)	274992	C(5,2)	113995	D(5,2)	101827	E(5,2)	584460
A(5,3)	253033	B(5,3)	276129	C(5,3)	240769	D(5,3)	163356	E(5,3)	465384
A(5,4)	374883	B(5,4)	295966	C(5,4)	211830	D(5,4)	113634	E(5,4)	169112

Edges	Weights	Edges	Weights	Edges	Weights	Edges	Weights	Edges	Weights
F(1,2)	67998	G(1,2)	42018	H(1,2)	18153	I(1,2)	8459	J(1,2)	4004
F(1,3)	213617	G(1,3)	301435	H(1,3)	181159	I(1,3)	231029	J(1,3)	166016
F(1,4)	194851	G(1,4)	185411	H(1,4)	215640	I(1,4)	202608	J(1,4)	115094
F(1,5)	165275	G(1,5)	165869	H(1,5)	325518	I(1,5)	89197	J(1,5)	57867
F(2,3)	106293	G(2,3)	67724	H(2,3)	44721	I(2,3)	36601	J(2,3)	13662
F(2,4)	233053	G(2,4)	271544	H(2,4)	284600	I(2,4)	218702	J(2,4)	191048
F(2,5)	211497	G(2,5)	242194	H(2,5)	296134	I(2,5)	190189	J(2,5)	152183
F(3,2)	200732	G(3,2)	183942	H(3,2)	210413	I(3,2)	200946	J(3,2)	118273
F(3,4)	103039	G(3,4)	54623	H(3,4)	88262	I(3,4)	13523	J(3,4)	10557
F(3,5)	209739	G(3,5)	126607	H(3,5)	342848	I(3,5)	168190	J(3,5)	81922
F(4,2)	180667	G(4,2)	170638	H(4,2)	328548	I(4,2)	89695	J(4,2)	58634
F(4,3)	213518	G(4,3)	241621	H(4,3)	289364	I(4,3)	191023	J(4,3)	150592
F(4,5)	35972	G(4,5)	18323	H(4,5)	23165	I(4,5)	1859	J(4,5)	2667
F(5,2)	159007	G(5,2)	167898	H(5,2)	366394	I(5,2)	136627	J(5,2)	84547
F(5,3)	198318	G(5,3)	241149	H(5,3)	301614	I(5,3)	183217	J(5,3)	160503
F(5,4)	162130	G(5,4)	124795	H(5,4)	339541	I(5,4)	130938	J(5,4)	63671

10 Conclusions

In this paper, we modeled the file recovery problem using a graph theoretic approach. We took into account the weight values of two directed edges connected to an edge to perform the file carving. We proposed two new algorithms to perform fragmented file recovery. The first algorithm, best path search, is suitable for files which have been fragmented into a small number of fragments. The second algorithm, high fragmentation path, is applicable in the cases where a file is fragmented into a large number of fragments. It introduces a trade-off between time and success rate of optimal path construction. This flexibility enables a user to adjust the settings according to his available resources. Analysis of the best path search technique reveals that it is much superior to brute force in complexity and at the same time, able to achieve accurate recovery. A sample of 10 files with their fragments were tested and the optimal carve is able to recover all of them back to their original correct state.

References

1. Leiserson, C.E.: Introduction to algorithms. MIT Press, Cambridge (2001)
2. da Gama Leito, H.C., Soltfi, J.: Automatic reassembly of irregular fragments. In: Univ. of Campinas, Tech. Rep. IC-98-06 (1998)
3. da Gama Leito, H.C., Soltfi, J.: A multiscale method for the reassembly of two-dimensional fragmented objects. IEEE Transections on Pattern Analysis and Machine Intelligence 24 (September 2002)
4. Shanmugasundaram, K., Memon, N.: Automatic reassembly of document fragments via context based statistical models. In: Proceedings of the 19th Annual Computer Security Applications Conference, p. 152 (2003)
5. Shanmugasundaram, K., Memon, N.: Automatic reassembly of document fragments via data compression. Presented at the 2nd Digital Forensics Research Workshop, Syracuse (July 2002)
6. Cohen, M.I.: Advanced jpeg carving. In: Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop, Article No.16 (2008)
7. Cohen, M.I.: Advanced carving techniques. Digital Investigation 4(supplement 1), 2–12 (2007)
8. Memon, N., Pal, A.: Automated reassembly of file fragmented images using greedy algorithms. IEEE Transactions on Image Processing, 385–393 (February 2006)
9. Sablatnig, R., Menard, C.: On finding archaeological fragment assemblies using a bottom-up design. In: Proc. of the 21st Workshop of the Austrain Association for Pattern Recognition Hallstatt, Austria, Oldenburg, Wien, Muenchen, pp. 203–207 (1997)
10. Garfinkel, S.: Carving contiguous and fragmented files with fast object validation. In: Proceedings of the 2007 Digital Forensics Research Workshop, DFRWS, Pittsburgh, PA (August 2007)
11. Martucci, S.A.: Reversible compression of hdtv images using median adaptive prediction and arithmetic coding. In: IEEE International Symposium on Circuits and Systems, pp. 1310–1313 (1990)

12. Kampel, M., Sablatnig, R., Costa, E.: 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), pp. 151–158 (2001)
13. Pal, A., Sencar, H.T., Memon, N.: Detecting file fragmentation point using sequential hypothesis testing. In: Proceedings of the Eighth Annual DFRWS Conference. Digital Investigation, vol. 5(supplement 1), pp. S2–S13 (September 2008)
14. Pal, A., Shanmugasundaram, K., Memon, N.: Automated reassembly of fragmented images. Presented at ICASSP (2003)
15. Stemmer, W.P.: DNA shuffling by random fragmentation and reassembly: in vitro recombination for molecular evolution. Proc. Natl. Acad. Sci. (October 25, 1994)