Automatic Reassembly of Document Fragments via Context Based Statistical Models

Kulesh Shanmugasundaram
kulesh@isis.poly.edu

Nasir Memon
memon@poly.edu

Department of Computer and Information Science
Polytechnic University
Brooklyn, NY 11201

Abstract

Reassembly of fragmented objects from a collection of randomly mixed fragments is a common problem in classical forensics. In this paper we address the digital forensic equivalent, i.e., reassembly of document fragments, using statistical modelling tools applied in data compression. We propose a general process model for automatically analyzing a collection fragments to reconstruct the original document by placing the fragments in proper order. Probabilities are assigned to the likelihood that two given fragments are adjacent in the original using context modelling techniques in data compression. The problem of finding the optimal ordering is shown to be equivalent to finding a maximum weight Hamiltonian path in a complete graph. Heuristics are designed and explored and implementation results provided which demonstrate the validity of the proposed technique.

1. Introduction

Reassembly of fragments of objects from a collection of randomly mixed fragments is a problem that arises in several applied disciplines, such as forensics, archaeology, and failure analysis. This problem is well studied in these disciplines and several tools have been developed to automate the tedious reassembly process [10]. The digital forensic equivalent of the problem –which we call *reassembling scattered documents*–, however, is yet to be explored.

Digital evidence by nature is easily scattered and a forensic analyst may come across scattered evidence in a variety of situations. A forensic analyst who comes across the problem of recovering deleted files often faces the difficult task of reassembling file fragments from a collection of randomly scattered data blocks on a storage media. This is especially true with the FAT16 and FAT32 filesystems, which due to the popularity of the Windows operating system, are perhaps still the most widely used file systems on personal computers. Furthermore, due to the ubiquitous presence of Windows and easier implementation considerations, the FAT file systems has been adopted in many consumer storage media devices, such as compact flash cards used in digital cameras and USB mini-storage devices. The FAT filesystem however is not very efficient in maintaining continuity of data blocks on the disk. Performance degradation due to file fragmentation is a common problem in many FAT systems. Due to fragmentation, when a file is stored data blocks could be scattered across the disk. Without adequate file table information it is difficult to put the fragments back together in their original order. Often critical file table information is lost because they are overwritten with new entries. In fact, the most widely used disk forensics tools like TCT[18], `dd` utility, The Sleuth Kit[7], and Encase[3] can recover data blocks from deleted files automatically. However, when the data blocks are not contiguous these tools cannot reassemble the blocks in the correct order to reproduce the original file without the proper file table entries. The job of reassembling these fragments is usually a tedious manual job carried out by a forensic analyst.

Another situation where a forensic analyst come across scattered evidence is the swap file. The system swap file is one of the critical areas where lot of useful forensic information can be gathered. The swap file contains critical information about the latest events that occurred on a computer. Therefore, reconstructing contents of the swap file is vital from a forensic standpoint. In order to achieve better performance, operating systems maintain swap file state and addressing information in page-tables stored only in volatile memory. When computers are secured for evidential purposes they are simply unplugged and sent to a forensic lab. Unfortunately contents of volatile memory are usually lost beyond recovery during evidence collection. Without the addressing information from the page-table it is dif-

ficult to rebuild contents off a swap file. Again, a forensic analyst is left with a collection of randomly scattered pages of memory.

One of the most popular and naive approach to hiding evidence is to store them in slack space in the filesystem. Files are assigned certain number of disk blocks for storage. However, not all files fit exactly into the allocated blocks. In most cases files end up using only a portion of their last block. The unused space in this last block is known as slack space. Modifying the contents of slack space does not affect the integrity of data stored in the filesystem because the read operation does not read data in slack space. A criminal can modify a file hiding program to choose the blocks on which files are hidden based on a sequence of numbers generated using a password. Knowing the password he can reconstruct the original document, whereas a forensic analyst is left with randomly mixed fragments of a document which will need to be reassembled.

Finally, ubiquitous networking and growing adoption of peer-to-peer systems give anyone easy access to computers around the world. There are many peer-to-peer systems which enable users to store data on a network of computers for easy, reliable access anytime, anywhere. *Freenet*[4], *Gnutella*[5] and *M-o-o-t*[12] are some of the better known systems used by millions of users around the world and many others, such as *OceanStore*[9], *Chord*[17] and *Pastry*[16], are in development at research laboratories. These systems are designed to provide reliable, distributed, and sometimes anonymous storage networks. A criminal can use these very systems to hide software tools and documents that might be useful for his prosecution, just as easily as any other user can save a file. Most peer-to-peer systems associate a unique key, either assigned by the user or generated automatically, with each document they store. Hence, a person can split a document into fragments and store each fragment in a peer-to-peer system using a sequence of secret phrases as keys, such that he can easily splice the fragments together knowing the proper sequence of secret phrases. For instance, in Freenet one can assign each fragment a unique URL. Since URLs are user friendly keywords it is easy to recall the proper sequence to retrieve and splice the fragments together. It is, however, difficult to reconstruct the original document without the knowledge of the proper sequence even if the keywords are known.

From the above discussion, it is clear that digital evidence can easily take a variety of forms and be scattered into hundreds of fragments making reassembly a daunting task for a human analyst. To address this problem we propose a general process model and present a specific solution to reassembling scattered evidence. Assuming that the necessary evidence is collected entirely, the proposed model has three steps:

1. **Preprocessing:** Encrypting or compressing digital evidence removes structural details that can assist an analyst in reassembling the evidence. During preprocessing, evidence has to be cryptanalyzed and transformed to its original form. Some cryptographic schemes derive their keys based on user passwords. Since users tend to choose dictionary based passwords it is quite feasible to attack the password and obtain the key regardless of the size of the key. Besides, brute force attacks on even some of the sophisticated cryptographic algorithms, such as DES, are shown to be feasible[6]. Note that a forensic analyst may not be too constrained on time, making cryptanalysis a feasible process.

2. **Collating:** Although, in this paper, we consider reassembling a single document, in reality evidence is usually a collection of mixed fragments of several documents of different types. To reassemble the evidence efficiently fragments that belong to a document must be grouped together. A hierarchical approach to collating can be used to effectively group similar fragments together. Fragments can be initially grouped by superficial characteristics, such as binary or plain-text document, and later sophisticated text-categorization techniques along with special knowledge about the fragments can be used to further refine the results.

3. **Reassembling:** The final step in the process is to either reassemble the document to its original form or to provide enough information about the original form to reduce the work of a forensic analyst. Ideally, we would like to obtain the proper sequence of fragments that resembles the original document. Even if the process identifies a small number of potential orderings, from which the forensic analyst can derive the proper ordering, it would result in considerable savings in time and effort to the analyst.

In this paper we focus on the final step, that is, reassembling a document given preprocessed fragments of that document. The rest of this paper is organized as follows: in the following section we describe the problem formally and introduce a general technique for document reassembly. Section 3 presents a specific realization of the general technique and initial experimental results and we conclude in section 4 with a discussion on future work.

2. The Fragment Reassembly Problem

In this section we formulate the document fragment reassembly problem in a more rigorous manner and describe a general approach for a solution to the problem.

2.1. Statement of the Problem

The problem of reassembly of scattered document can be stated as follows: suppose we have a set $\{A_0, A_1 \ldots A_n\}$ of fragments of a document 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 document A. We assume fragments are recovered without loss of data, that is, concatenation of fragments in the proper order yields the original document intact.

Note that in order to determine the correct fragment reordering, we need to identify fragment pairs that are adjacent in the original document. One simple technique to do this, is to use a dictionary of the underlying language of the document. With this approach, a fragment A_j would be considered to be a likely candidate fragment to follow A_i if the word that straddles the boundaries of A_i and A_j is found in the dictionary. For example, suppose fragment A_i ends with the phrase "*The quick bro*" and fragment A_j, A_k begin with the phrase "*wn fox jumps...*" and "*over lazy dog...*" respectively. It is clear that A_j is a better candidate to follow A_i than A_k as the word that straddles the boundary of fragments A_i, A_j forms a dictionary word, whereas the word that straddles the boundary of fragments A_i, A_k does not.

However, a dictionary-based approach is language specific and it is therefore not feasible for the variety of documents a forensic analyst may come across in the field. Furthermore, for non-textual files, like executables, a dictionary may not be readily available or easy to construct. Finally, if more than one dictionary matches are found then how does a forensic analyst select between the two?

To quantify the likelihood of adjacency a linguist may assign *candidate probabilities* $C_{(i,j)}$, representing the probability that the fragment A_j follows A_i, based on syntactic and semantic analysis for each pair of fragments. Once these probabilities are assigned, the permutation of the fragments that leads to correct reassembly, among all possible permutations, is likely to maximize the product of candidate probabilities of adjacent fragments. This observation gives us a technique to identify the correct reassembly with high probability. More formally, we want to compute the permutation π such that the value

$$\prod_{i=0}^{n-1} C(\pi(i), \pi(i+1)) \tag{1}$$

is maximized over all possible permutations π of degree n. This permutation is most likely to be the one that leads to correct reconstruction of the document. Note that maximizing the product in equation (1) is equivalent to maximizing the sum

$$\sum_{i=0}^{n-1} -\log C(\pi(i), \pi(i+1)) \tag{2}$$

The problem of finding a permutation that maximizes the sum in equation (2) can also be abstracted as a graph problem. To do this we take the set of all candidate probabilities ($C_{i,j}$) to form an adjacency matrix of a complete weighted graph of n vertices, where vertex i represents fragment i and the edge weights quantify the candidate probability of two corresponding fragments being adjacent. The proper sequence π is a path in this graph that traverses all the nodes and maximizes the sum of candidate probabilities along that path. The problem of finding this path is equivalent to finding a maximum weight Hamiltonian path in a complete graph (See Figure 1) and the optimum solution to the problem turns out to be intractable[2]. However there are many heuristics known in the literature and we employ one such heuristic as discussed in Section 3.2.

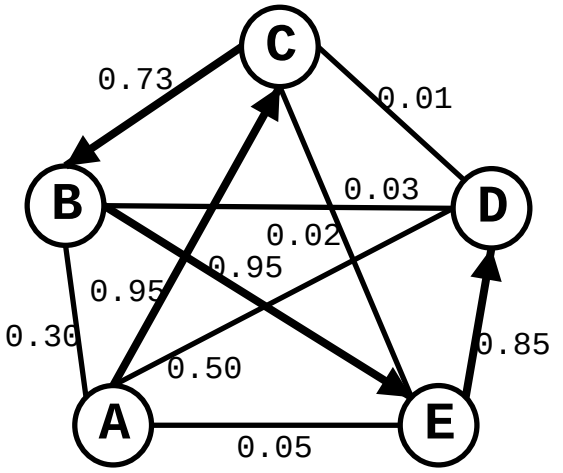

Figure 1. A Complete Graph of Five Fragments & Hamiltonian Path (ACBED) that Maximizes the Weight ($0.95 + 0.73 + 0.95 + 0.85 = 3.48$**)**

It should be noted that the optimal solution may not necessarily result in reconstruction of the original document. However, if candidate probabilities 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 document reassembly tool to present to the forensic analyst a small number of most likely reorderings, and based on which the correct reordering can be manually arrived at. The question that remains is how do we assign candidate probabilities for pair of fragments being adjacent, in an efficient and meaningful

manner? We address this question in the following subsection by using context-based statistical models.

2.2. Context-Based Statistical Models

The model for assigning candidate probabilities for a pair of fragments needs to be independent of language and format. Data compression literature explores many powerful statistical modelling techniques to build data models to effectively compress data. The modern data compression paradigm, first presented in [14], divides the compression process into two components: *modelling* and *coding*. Modelling is the process of constructing statistical representations of input data and coding is the process of mapping information generated by statistical representations to bit sequences to produce compressed data. Modelling represents the critical step in a data compression system and over the years a variety of modelling techniques have been devised.

Modelling techniques essentially build a statistical model of the input to predict the occurrence probabilities of symbols. Specifically, given a realization $x_1, x_2, \ldots, x_m$ of a finite sequence of random variables $X_1, X_2, \ldots, X_m$ over a discrete alphabet $\mathcal{A}$, a model essentially assigns a conditional probability mass function for the current symbol (event) based on previously processed symbols [15]. In *on-line* applications, the sequence $x_1, x_2, \ldots, x_m$ is processed in a *sequential* manner, with the symbol x_i being encoded immediately before the symbol x_{i+1}. In this case we need to estimate the distributions

$$p(X_{i+1} = x_{i+1} | X_1 = x_1, \ldots, X_i = x_i), \;\; 1 \leq i < m \quad (3)$$

since the average number of bits needed to encode the realization $x_1, \ldots, x_m$ online is bounded below by

$$\sum_{i=0}^{m-1} H(X_{i+1} | X_1, X_2, \ldots, X_i). \quad (4)$$

where $H(\cdot)$ is the Shannon (conditional) entropy function. Coding techniques that can achieve rates close to the optimal are known [14].

The conditional distribution for the r.v. X_{i+1} in Equation (3) is best estimated (in principle) by using the entire past sequence, $X_j, \; j = 1 \ldots i$. However, in practice, knowledge about the statistics of the source sequence is nowhere as complete. Practical compression techniques usually impose some structural limitations on the source sequence to arrive at a model that can realistically represent the sequence. A particularly popular model that has been widely used is a *context model* [13].

A context model assumes that the distribution of the current symbol only depends on some *limited context* in which it occurs. In particular, associated with a context model is a finite set of *contexts* or *conditioning events* $\mathcal{C}$ along with a *context determining rule* or function that maps the first i symbols ($0 \leq i < N^2$) of the source sequence to some context $C \in \mathcal{C}$. The symbol x_{i+1} is then said to appear in the context C. An $n^{th}-$order context model uses n previous symbols in order to estimate the probability of the next symbol. However, a context model may be a blended model in which it incorporates probability estimation based on several different orders.

Usually the number of distinct contexts (*i.e.* the size of the set $\mathcal{C}$) is much smaller than the length of the source sequence. Associated with each context C is a probability distribution $p(x|C)$ that is used to encode pixel X_j when its context is C. This pdf can be estimated by maintaining counts of symbol occurrences within each context or by estimating the parameters of an assumed pdf.

Context models are especially intuitive for text compression as the probability of occurrence of a letter clearly depends on its context, i.e. the immediately preceding letters. For example, the probability of symbol 'u' in a document written in English may be $1/10$. However, if the preceding symbol was 'q' then probability of 'u' can go up to $9/10$, because 'u' follows 'q' in most English words. The idea of context consisting of few previous symbols is intuitive for models representing natural languages. Empirical results, however, show context modelling provides better models for compressing not only natural language but also a variety of other data formats including images, and executables [1].

2.3. Our Approach

Now we have all the pieces required to describe a general approach to reassembling document fragments. First we build an $n-$order context model for a given document by accumulating the context models of individual fragments into a single model. Using this model, for any ordered pair of fragments consider a sliding window of size n. Place the window at the end of the first fragment and slide the window one position at a time into next fragment, at each position estimating the probability of the upcoming character $(n+1)$ by using the characters in the window as the current context and using the context model obtained from the total pool of fragments for the purpose of probability estimation. (See Figure 2) Continuing this process for d subsequent characters, where $d \leq n$, gives the candidate probability for the fragment pair as given in equation (1). Repeating the process for all fragments results in a complete graph where the edges quantify candidate probabilities of the adjacency of corresponding nodes in the original document. We then use a heuristic solution to compute a small number of near-optimal reorderings of the fragments. The actual re-

ordering is likely to be contained in this set or at worst can be easily derived from this set by a forensic analyst.

Note that even if the analyst can identify a subsequence of any of these reorderings to be correct, she can combine the fragments that belong to the subsequence to form a unit fragment and repeat the process. At every iteration if the analyst can successfully find correct subsequences which can be merged, then the process will eventually converge to the original document.

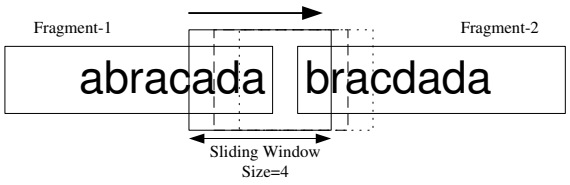

Figure 2. Sliding Window of Size 4

3. Implementation & Experiments

In this section we describe an implementation of our approach which employs a well known context modelling technique known as Prediction by Partial Matching (PPM) to build a context model and compute candidate probabilities of the possible adjacency of two document fragments. Also we use an alpha-beta pruning heuristic to solve the Hamiltonian path problem and compute a small set of near-optimal candidate reorderings. We present experimental results with different types of data which demonstrate the validity of our approach.

3.1. Prediction by Partial Matching

Prediction by partial matching or PPM, is a finite order context modelling technique first introduced in 1984 by Cleary & Witten and has become a benchmark for lossless data compression techniques[1]. PPM employs a suite of fixed-order context models, from 0 up to some predetermined maximum k, to predict the upcoming characters. For each $i-$order context model statistics are kept of all symbols that have followed every $i-$length subsequence observed so far in the input and number of times that each has occurred. Prediction probabilities are computed from these statistics in the following way: From each model, $0-$order to $k-$order, a separate predicted probability distribution is obtained and effectively combined into a single one. The largest-order model is the one, by default, used to initially predict the symbol. However, if a novel symbol is encountered an "escape" symbol is returned and a smaller model is used for prediction. The process continues until one of the fixed-order models predicts the upcoming symbol. To ensure this process terminates a model is assumed to be present (below $0-$order) with all symbols in the coding alphabet. (Note that we don't need this order because the model already knows the entire alphabet and will never encounter a novel symbol.) This mechanism effectively blends the prediction probabilities of different order models together in a proportion that depends on the values used for escape probabilities.

In our model, each fragment of the document is individually processed by PPM and the resulting statistics are combined to form a single model. Assuming there is no data loss during evidence collection, we believe the resulting model is a good statistical representation of the original document. Suppose we used an $i-$order PPM model then we can use a sliding window of size i or less and predict the upcoming symbol as discussed in Section 2.3. Prediction probabilities of each model are combined into a single probability using PPMC method as described in[11]. The resulting probability is the candidate probability of adjacency for a pair of fragments.

3.2. Tree Pruning

As discussed in Section 2.2, the problem of finding the optimal solution to the Hamiltonian path is intractable. Assuming we know the first fragment of a document we can represent all the paths in the underlying graph by a tree (See Figure 3). The assumption that we can identify the first fragment is practical because most files have headers at the beginning that can uniquely identify the filetype.

Finding the optimal solution in this tree simply means examining each path in the tree in Figure 3 and finding the one that maximizes the sum of candidate probabilities along that path. However, as we can see, the tree expands exponentially as the number of levels increase. In our case the number of levels equals the number of fragments, which can run into the hundreds. One approach to make the problem tractable is to prune this tree at every stage of expansion by removing paths that do not appear to be promising. Another way to look at this is that we prune the tree at every level by keeping only the most promising paths such that in the end we obtain a set of near optimal heuristic solutions. The pruning approach we adopt uses this approach and is adopted from α-β pruning used in game theory[8].

By pruning we try to avoid examining paths that we believe may not contribute enough to our solution. A naive approach to pruning the tree is to choose a node with maximum candidate probability at each level. However, such a greedy method could lead to poor solutions. Nevertheless, the greedy method can be extended to look not only at current level but also at β levels deep and choose a node at cur-

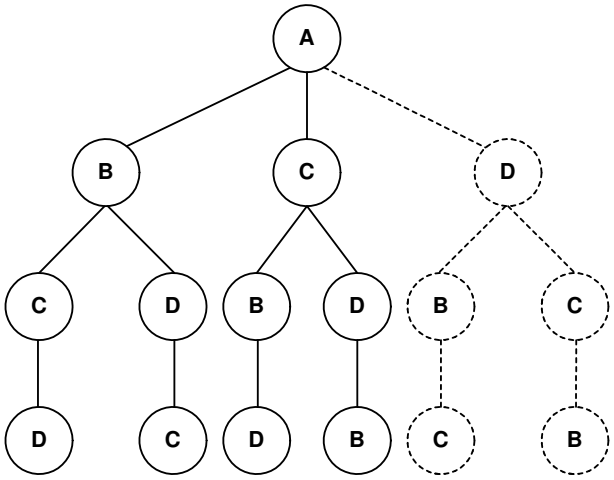

Figure 3. A Tree of Four Fragments with Path (A, D) Pruned

rent level that maximizes the sum of candidate probabilities. In addition, instead of choosing a single node at each level, which limits our results to a single sequence, we choose α best matches at each level resulting in α best sequences. We employed this α-β tree pruning approach in our implementation and report the results obtained below.

3.3. Experiments & Results

This section presents experimental results and discussion of the results. We used the following collection of documents in our experiments:

Type	Samples
Log Files	Log, history files of various users
Source Code	C, C++, Java, Lisp source code
Executables	Executable, object code
Binary Files	MS Office documents, PDF
Unformatted Text	Unformatted plain-text, chat transcripts
Random Text	Encrypted, compressed files

Table 1. Documents Used in the Experiments

Each document in the collection was randomly split into several pieces (100 - 200) and a context model was built for each document. This model was used along with pruning to identify the most promising reorderings. Accuracy of reassembly was measured by the number of adjacent fragments in the reassembled documents that are in fact adjacent in the original document. Figure 4 presents the average accuracy of reassembly process for each document type and Figure 5 presents the average compression ratio for each type, which is an indicator of the structure of documents.

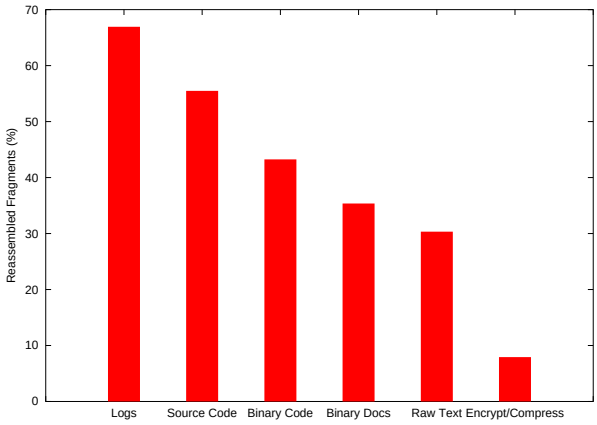

Figure 4. Average Reassembly of Fragments in a Single Pass

Log files and other operating system related files are reassembled more accurately as they have more structure and data is often repeated in a predictable manner. Likewise, source code has standard keywords and when broken along these keywords a matching fragment is easily found. Binary code and binary documents, however, have less predictable patterns and most binary files seem to have 'file holes', where large regions are filled with '0's. When two or more fragments are split along these file holes they have multiple candidates for the adjacent fragment as their candidate probabilities are uniformly distributed. In this case, instead of breaking ties arbitrarily fragments are chosen by increasing the value of β, which helps choose the most likely fragment among the competing fragments by looking further down the path. In addition to file holes, some binary documents have compressed regions, such as inline images, which further affects their accurate reassembly. Unformatted plain-text and chat transcripts also proved to be difficult to reassemble since the transcripts contain unpredictable words and fragments are split along large portions of empty spaces. Figure 5 shows the average compression ratio of sample document types and as we compare this to Figure 4 it is clear that more structure a document has better the accuracy of reassembly.

Table 2 presents most accurate reassembly in top-n, that is $(\alpha \in \{5, 10, 15, 20\})$, candidate reorderings.

At first sight the numbers in Table 2 may appear to be low. However, it should be noted that a forensic analyst can examine the top α potential orderings by our system and identify proper subsequences within them, That is, subse-

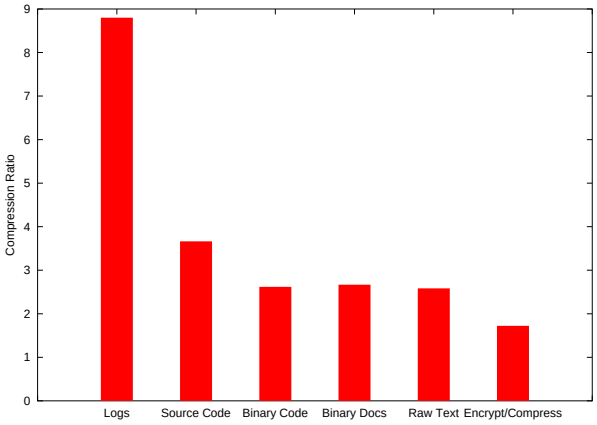

Figure 5. Compression Ratio of Various Document Types

Type	Top 5	Top 10	Top 15	Top 20
Log Files	57.7%	58.0%	58.7%	68.0%
Executables	30.0%	30.7%	33.4%	33.4%
Binary Files	23.4%	24.6%	28.4%	28.4%
Unformatted	26.4%	28.3%	29.0%	31.0%

Table 2. Reassembled Document Fragments in Top $\{5, 10, 15, 20\}$ Candidates in a Single Pass

Type	Iterations
Log Files	4
Source Code	6
Executables	9
Binary Files	10
Unformatted	10

Table 3. Iterations Required to Reconstruct the Entire Document

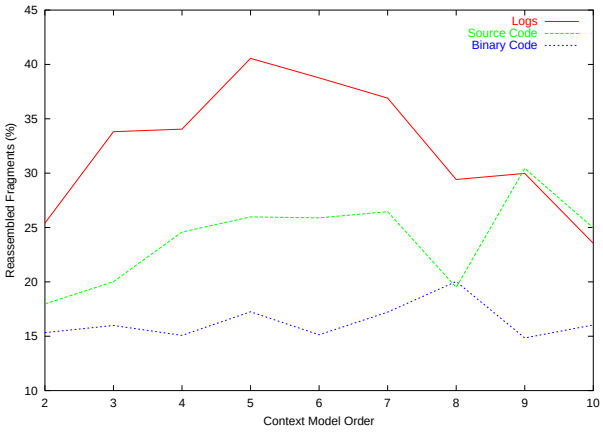

Figure 6. Influence of Context Orders in Reassembly of Various Document Types

quences that correspond to proper reassembly. The fragments in these subsequences can then be recombined into unit fragments and the entire reassembly process reiterated. This iteration process will eventually converge on to the proper reordering with much less effort than if she were to perform the entire task manually. The following table lists the average number of iterations required to reconstruct various document types. It can be seen that even with unformatted files and binary files, about 10 iterations are sufficient to converge to the correct reordering. For a data set with more than a hundred fragments for each file, this is a reasonably small effort compared to any manual analysis.

In general smaller order contexts (3 to 5) performed well for various document types. Figure 6 illustrates the optimal context orders for log files, source code, and executable files. Increasing the size of the context had no effect in most cases.

When candidate probabilities are distributed properly, that is a fragment's candidate probability to another is significantly larger than for rest of the fragments, increasing α and β yields better results. When candidate probabilities are uniformly distributed, that is each fragment is equally probable candidate to be adjacent fragment to others, tweaking α, β has no influence on the results. We believe preprocessing heuristics are s vital part of the process and will further enhance the accuracy of reassembly. Furthermore, domain knowledge of binary documents and code, such as valid instructions or valid tags, and can be incorporated into the model to more accurately compute candidate probabilities.

4. Conclusion

Digital evidence by nature is easily scattered and a forensic analyst may come across scattered evidence in a variety of situations. For example, a forensic analyst who comes across the problem of recovering deleted files often faces the difficult task of reassembling file fragments from a collection of randomly scattered data blocks on a storage media. In this paper, we introduce a general framework for reassembling scattered evidence using context based statistical models. We formulate the problem of reconstructing the most likely reordering of the fragments as a graph problem and showed that computing the optimal solution involves solving the maximum weight Hamiltonian Path problem

which is known to be intractable. We then proposed a tree-pruning heuristic to compute a set of near-optimal solutions. We implemented our framework using PPM, a well known context modelling technique in the data compression literature, and show that our approach gives promising results.

Further investigation is necessary to establish effective preprocessing heuristics for various document types. Although we would like to have the reassembly process to be independent of document types incorporating meta information, such as the syntax of a language or instruction set of a program, about a certain document type into another level of abstraction will help us reassemble the document more accurately. Since images are often implicated in criminal investigations than other types of documents we are currently investigating reassembling of various image formats from fragments. Also, we are planning to investigate methods to collate fragments of documents from mixed fragments of several document types using text classification methods.

References

[1] J. G. Cleary and W. J. Teahan. Unbounded length context for ppm. *The Computer Journal*, 1997.

[2] T. H. Cormen and e. a. Leiserson C. E. Introduction to algorithms. *MIT Press*, 2001.

[3] G. S. I. Encase. http://www.encase.com/.

[4] Freenet. http://freenetproject.org/.

[5] Gnutella. http://gnutella.wego.com/.

[6] I. Hamer and P. Chow. Des cracking on the transmogrifier 2a. *Cryptographic Hardware and Embedded Systems, LNCS 1717, Springer-Verlag,*, pages 13–24, 1999.

[7] T. S. Kit. http://www.sleuthkit.org/.

[8] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, pages 293–326, 1975.

[9] J. Kubiatowicz and D. e. a. Bindel. Oceanstore: An architecture for global-scale persistent storage. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[10] H. C. G. Leitao and J. Stolfi. A multi-scale mehtod for the re-assembly of fragmented objects. *Proc. British Machine Vision Conference - BMVC 2000*, 2:705–714, 2000.

[11] A. Moffat. Implementing the ppm data compression scheme. *IEEE Transactions on Communications*, 1990.

[12] M. o-o t. http://www.m-o-o-t.org/.

[13] J. J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29(5):656–664, 1983.

[14] J. J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM J. Res. Dev.*, 23(2):149–162, 1979.

[15] J. J. Rissanen and G. G. Langdon. Universal modelling and coding. *IEEE Transactions on Information Theory*, 1981.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001.

[17] I. Stoica and R. e. a. Morris. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM 2001*, pages 149–160, 2001.

[18] T. C. T. (TCT). http://www.porcupine.org/forensics/tct.html.