Predicting the Types of File Fragments

By

William Calhoun, Drue Coles

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2008 USA

Baltimore, MD (Aug 11th - 13th)

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

Predicting the types of file fragments

William C. Calhoun[1], *Drue Coles*[*,1]

Department of Mathematics, Computer Science and Statistics, Bloomsburg University of Pennsylvania, Bloomsburg, PA 17815, USA

Keywords:
File types
File fragments
Predicting types
Linear discriminant
Longest common subsequence
Entropy

ABSTRACT

A problem that arises in computer forensics is to determine the type of a file fragment. An extension to the file name indicating the type is stored in the disk directory, but when a file is deleted, the entry for the file in the directory may be overwritten. This problem is easily solved when the fragment includes the initial header, which contains explicit type-identifying information, but it is more difficult to determine the type of a fragment from the middle of a file. We investigate two algorithms for predicting the type of a fragment: one based on Fisher's linear discriminant and the other based on longest common subsequences of the fragment with various sets of test files. We test the ability of the algorithms to predict a variety of common file types. Algorithms of this kind may be useful in designing the next generation of *file-carvers* – programs that reconstruct files when directory information is lost or deleted. These methods may also be useful in designing virus scanners, firewalls and search engines to find files that are similar to a given file.

1. Introduction

Computer forensic analysts must often reconstruct a file from the fragments remaining after the user has attempted to delete the file. Typically, parts of it will have been overwritten by newer files. The analyst may not even know the file type of a given fragment. There is a simple and reliable way to determine the type: check the header. This is made all the easier by the trend to open source formats, but even with proprietary formats it is possible to find explicit type-identifying information near the beginning of the file.

We are interested in the situation where parts of the meta-data have been lost or corrupted. Even if the entire header is lost, we can still hope to reliably predict the file type because many formats embed meta-data throughout the file. A PNG file, for example, starts with an 8-byte signature, part of which exists solely for the purpose of file type identification with a text editor, and following the header is a sequence of segments containing information about the image, each segment with its own header. This is followed by the actual data and some redundancy for error correction (Roelofs, 1999).

Current computer forensic software may not be able to correctly identify the type of a fragment without the header intact. In this paper, we investigate two algorithms for predicting the type of a fragment. One is based on Fisher's linear discriminant and the other based on longest common subsequences of the fragment with various sets of test files. The results are promising, although work is needed to improve the accuracy and efficiency of our software. Ultimately, we expect algorithms of this kind to be useful in designing the next generation of *file-carvers* – programs that reconstruct files when directory information is lost or deleted (Garfinkel, 2007; Richard and Roussev, 2005). They may also be useful in designing virus scanners, firewalls and search engines to find files that are similar to a given file.

Previous research in identification of file types includes the use of file "fingerprints" by McDaniel and Heydari (2003) and the similar "fileprints" used by Li et al. (2005). In this approach the "fileprint" of a collection of files is the histogram of byte frequencies together with their variance. Centroid (average) fileprints are found for collections of files of each type. The type of a file can then be predicted by matching the histogram

* *Corresponding author.*
E-mail address: dcoles@bloomn.edu (D. Coles).

[1] Both authors were supported by a Bloomsburg University Research and Development Grant during the summer of 2006, when this project was initiated.
1742-2876/$ – see front matter © 2008 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2008.05.005

of byte frequencies for the "unknown" file to the nearest (in Mahalanobis distance) centroid fileprint. This approach is related to the Fisher linear discriminant we use, but the experiments conducted in these papers are limited to type identification for full files or truncated files containing the header. Our interest is in identifying the types of file fragments not containing the header.

Karresand and Shahmehri (2006a,b) developed the "Oscar" method for identifying the types of file fragments. This approach also uses centroid fileprints, but different metrics are used. The algorithm has been optimized for JPG files, using information about specific marker byte pairs in the data part of JPG pictures. In tests conducted by the authors, Oscar identified JPG fragments with a 99.2% detection rate and no false positives. Although Oscar is applicable to any file type in principle, the excellent results reported in these papers are limited to JPG fragments and are obtained using special information about JPG file-structure. Karresand and Shahmehri also use the rate of change of the bytes in addition to the frequency. They define the rate of change to be the absolute value of the difference between two consecutive byte values. This allows the ordering of the bytes to be taken into consideration. We have used a related idea by calculating the correlation between successive bytes as described below.

A recent paper by Veenman (2007) is closest to our approach. The Fisher linear discriminant is applied to the fileprint, entropy and a measure based on Kolmogorov complexity. The main difference in our approach is the application of the Fisher linear discriminant to several different statistics and our use of the longest common subsequence algorithm. It is difficult to compare our results directly to Veenman's since we conducted a different set of experiments to test our approach than Veenman did. However, we will attempt to make some comparisons after describing our approach and results.

2. Type prediction with the linear discriminant

The *frequency* (or *relative frequency*) of a symbol in a string is the number of occurrences of the symbol divided by the length of the string. Computer forensic analysts use the distribution of frequencies of byte values in a file fragment to identify its type (Shannon, 2004). Particularly useful is the frequency of ASCII codes. Clearly, ASCII frequency is high in text files and lower in non-text files. Another useful statistic is the *entropy* of the file fragment. In information theory, the *Shannon entropy* (or *information entropy*) of a sequence of symbols is a measure of how much information it contains (Shannon, 1948). For example, long strings of zeros have low entropy, since they contain very little information. Random sequences, on the other hand, have high entropy. Text files have low entropy since much of the information in the file is redundant. Compressed and encrypted files have high entropy. Entropy is calculated as the sum of $-P_v \log_2 P_v$ over all byte values v, where P_v is the frequency of v in the fragment. One method we have used to predict file types is to create a mathematical classification model based on the ASCII frequency, entropy, and other statistics.

Fisher's linear discriminant is a statistical method used to classify individuals into groups (Huberty and Olejnik, 2006; James, 1985). There is an initial training phase in which data from individuals belonging to known groups are used to develop a classification model. The classification model is a set of linear functions, one for each group. Data from individuals whose group membership is unknown can then be entered into the classification model. The model predicts the group to which the individual belongs according to the function that returns the highest value. Our software calculates the linear discriminant based on statistics calculated from sample files. We have experimented with several different statistics and combinations of statistics including: frequency of ASCII codes, entropy, modes, mean, standard deviation, and correlation between adjacent bytes. These statistics are listed in Table 1. (We omit discussion of norming factors used to put all of the statistics in the range between 0 and 1.)

If is a familiar fact that a linear regression line is the line that best fits a given set of data. Fisher's linear discriminant is similarly optimal. Under certain assumptions about the data, Fisher's linear discriminant can be shown to be the linear discriminant that minimizes the total classification error. For optimality, it is sufficient (but not necessary) that the data come from a multivariate normal distribution and that the groups have identical covariance matrices. When the linear discriminant is applied to data that do not satisfy these assumptions the results may be sub-optimal. However, Fisher's linear discriminant often remains useful when the data approximately satisfy the assumptions. In that case, the prediction functions are often close to optimal. Our results are clearly not optimal in some cases, particularly when we use a large number of statistics to make the predictions. The assumptions needed for the linear discriminant to work well are less likely to hold when we use a larger number of statistics. Furthermore, the linear discriminant algorithm involves inverting an n by n matrix, where n is the number of statistics. Thus, increasing the number of statistics may also lead to increased inaccuracy due to floating point computation errors.

We have also experimented using a genetic algorithm in place of the linear discriminant to find near-optimal classification functions (Mitchell, 1996). In some cases, the genetic algorithm provides better classification functions than does the linear discriminant. However, the genetic algorithm is much slower than the linear discriminant algorithm and is, moreover, nondeterministic. The genetic algorithm begins with a population of randomly generated prediction functions, and at successive stages the functions "complete" to "reproduce" on the basis of their accuracy in predicting the types of given test fragments. Over time the population evolves (in theory) to include better and better predictors, replacing poor ones in the process. We find that the performance of the genetic algorithm varies significantly from one trial to the next. The results described in this paper were obtained using the linear discriminant instead.

3. Type prediction with longest common substrings and subsequences

Our next idea for file type prediction is based on the idea that two files of the same type will probably have longer substrings in common than would files of different types. Our results

Table 1 – Statistics used in calculating linear discriminant

Statistic	Definition
Low	Frequency of bytes in the fragment with value v, where $0 \leq v < 32$.
Ascii	Frequency of bytes in the fragment with value v, where $32 \leq v < 128$.
High	Frequency of bytes in the fragment with value v, where $192 \leq v < 256$.
Entropy	Sum of $-P_v \log_2 P_v$ over all byte values v, where P_v is the frequency of v in the fragment.
ModesFreq	The sum of the four highest byte frequencies.
Mu	Mean byte value.
Sd	Standard deviation of the byte values.
CorNext	Correlation of values of the nth and $n+1$st bytes.
SdFreq	Standard deviation of the byte frequencies.
CorNextFreq	Correlation of the frequencies of byte values m and $m+1$.
ALCSStrA	Average length of the longest common substring of the fragment with the type A training files.
ALCSStrB	Average length of the longest common substring of the fragment with the type B training files.
ALCSSeqA	Average length of the longest common subsequence of the fragment with the type A training files.
ALCSSeqB	Average length of the longest common subsequence of the fragment with the type B training files.
LongerSStrA	1 If *ALCSStrA* > *ALCSStrB*, 0 otherwise.
LongerSSeqA	1 If *ALCSSeqA* > *ALCSSeqB*, 0 otherwise.
A–E	*Ascii*, *Entropy*
A–E–CNF	*Ascii*, *Entropy*, *CorNextFreq*
A–E–H–S–CN–SF	*Ascii*, *Entropy*, *High*, *Sd*, *CorNext*, *SdFreq*
A–E–L–H–MF–SF	*Ascii*, *Entropy*, *Low*, *High*, *ModesFreq*, *SdFreq*
A–E–ALCSSA–ALCSSB	*Ascii*, *Entropy*, *ALCSSeqA*, *ALCSSeqB*

The statistics used are listed and defined in the top rows of the table. The statistics in the middle rows of the table are calculated using the longest common substring and longest common subsequence algorithms. The bottom rows give acronyms for combinations of statistics used as data for the linear discriminant.

show that this is true even when each file is missing part of its meta-data. We extend the idea to consider common subsequences for enhanced prediction capabilities. A subsequence of a byte stream is any sequence of bytes obtained by deletions from the original. The intuition for considering subsequences is that several small substrings of meta-data may be separated by long strings of actual data, and by considering subsequences we are in effect eliminating the useless (for type prediction) data and concatenating the useful parts into a single string.

Suppose that we have two files of length m and n, with $m < n$. The naive approach to computing their longest common subsequence is to look in the longer file for all 2^m subsequences of the shorter file, but this takes on the order of $n2^m$ byte comparisons, requiring millions of years even for modest file lengths. We use the standard dynamic programming approach to this problem, which runs in time roughly mn – very fast. We also use dynamic programming to compute longest common substrings with the same runtime complexity (Cormen et al., 2002).

4. Testing the algorithms

We tested our algorithms by using them to predict the file types of a set of file fragments. Although the file fragments are taken from files of known types, this type is not supplied to the algorithms. The rate at which an algorithm correctly predicts the types of the file fragments gives us a measure of its performance. We recorded the total percentage of correct file type predictions, for 100 files, 50 files of one type (type A) and 50 of another type (type B). We considered two different cases. One simulates the situation in which a small section at the start of the file has been lost (or deliberately deleted). We removed the first 128 bytes of the files and used the next 896 bytes. The results are shown in Table 2. The other case simulates the situation in which the second sector of a file is recovered but not the first. Thus the unknown file fragments are 512 bytes long and were selected from files of the given type with an offset of 512 bytes from the start of the file. The results are shown in Table 3.

Not surprisingly, our algorithms can accurately distinguish between obviously different formats. For instance, they can easily distinguish JPG fragments from fragments of Excel or text documents. In fact the prediction can be made with 100% accuracy on the basis of the entropy alone. Since the use of ASCII frequency and entropy to identify file types is already well-known to computer forensic analysts, we have focused our testing on cases where ASCII frequency and entropy do not provide enough information to reliably distinguish the types. For simplicity, all of our tests are two-way tests involving 50 file fragments of type A and 50 file fragments of type B. We used the following graphics file formats: JPEG (JPG), bitmap (BMP) and Graphics Interchange Format (GIF). We also used the Adobe Portable Document Format (PDF), which combines text and graphics.

Table 2 shows the results in the case where the first 128 bytes of a 1024 byte file fragment are removed. In four of the six pairwise tests, the best rate of correct prediction was obtained by using the longest common subsequence technique either by itself or combined with the ASCII and entropy statistics in the linear discriminant. The combined approach resulted in 100% accuracy in distinguishing JPGs from GIFs, 92% accuracy in distinguishing JPGs from BMPs and 87%

Table 2 – Results with first 128 bytes removed from a 1024 byte fragment

Statistic/types	JPG vs. PDF	JPG vs. GIF	JPG vs. BMP	PDF vs. GIF	PDF vs. BMP	GIF vs. BMP	AVG
Ascii	86	59	77	86	82*	72	77
Entropy	78	69	82*	76	64	80	74.8
Low	99*	89	70	72	74	69	78.8*
High	83	69	75	77	69	66	73.2
ModesFreq	78	66	82*	75	63	77	73.5
Sd	86	80	51	87*	78	54	72.7
CorNext	78	86*	81	78	55	81*	76.5
SdFreq	78	66	50	73	65	78	68.3
CorNextFreq	86	73	78	63	61	67	71.3
A–E	89	69	82	87	80	79	81
A–E–CNF	50	73	82	87	79	81	75.3
A–E–H–S–CN–SF	98*	50	86*	89*	84*	83*	81.7
A–E–L–H–MF–SF	98*	93*	85	89*	82	83*	88.3*
LongerSStrA	93	88	78	88	86	72	84.2
LongerSSeqA	94*	87	74	92*	79	81*	84.5*
A–E–ALCSSA–ALCSSB	50	100*	92*	87	87*	80	82.7

Offset = 128, File Fragment length = 896. The percentage of correct type predictions is shown for individual statistics and combinations of statistics. In each case the problem is to distinguish files of the two file types indicated at the top of the column. The last column, labeled AVG, shows the average of the percentages of correct predictions in the preceding columns. The highest percentage(s) in each section of a column are marked with an asterisk. The highest percentage in the column is displayed in italics face.

accuracy in distinguishing PDFs from BMPs. The longest common subsequence technique worked slightly better without ASCII and entropy in distinguishing PDFs from GIFs, with a 92% rate of correct prediction. Interestingly, JPGs and PDFs were distinguished with 99% accuracy on the basis of the Low frequency alone. In the attempts to distinguish GIFs from BMPs, the linear discriminant approach worked best. Two different combinations of statistics provided 83% accuracy. Of course, it would be desirable to have a method that works reasonably well for all file types. For the file types used in this experiment the linear discriminant using the statistics *Ascii*, *Entropy*, *Low*, *High*, *ModesFreq* and *SdFreq* worked best with an average rate of correct prediction of 88.3%. For comparison, the average rate of correct prediction using only *Ascii* and *Entropy* was 81%.

Table 3 shows the results in the case where the first 512 bytes of a 1024 byte file fragment are removed. As one would expect, the rates of prediction are generally lower with more bytes removed. However, in most cases the rate in Table 3 only differs by a few percentage points from the rate in Table 2 (and is sometimes higher). The highest rate of correct prediction was 95%, obtained when using the longest common substring technique to distinguish JPGs from PDFs. In three of the other five pairwise tests, the highest rate of correct prediction

Table 3 – Results with first 512 bytes removed from a 1024 byte fragment

Statistic/types	JPG vs. PDF	JPG vs. GIF	JPG vs. BMP	PDF vs. GIF	PDF vs. BMP	GIF vs. BMP	AVG
Ascii	82*	68	86	82*	77*	76	78.5*
Entropy	79	70	70	76	69	80	74.0
Low	72	69	72	72	75	68	71.3
High	79	54	70	71	77*	63	69
ModesFreq	80	74	82	75	67	78	76
Sd	82*	69	50	81	75	54	68.5
CorNext	79	69	80	78	56	80*	60.3
SdFreq	77	64	78	71	67	77	72.3
CorNextFreq	80	77*	87*	69	70	80	77.2
A–E	82	74	82	82	87	80	82.2
A–E–CNF	83	52	91*	70	87	80	77.2
A–E–H–S–CN–SF	84*	80*	87	82	85	86*	84
A–E–L–H–MF–SF	80	79	87	84*	90*	85	84.2*
LongerSStrA	95*	75	78	72	83	68	78.5
LongerSSeqA	91	85*	75	90*	92*	83*	86*
A–E–ALCSSA–ALCSSB	50	83	83*	84	91	52	73.8

Offset = 512, File fragment length = 512. The percentage of correct type predictions is shown for individual statistics and combinations of statistics. In each case the problem is to distinguish files of the two file types indicated at the top of the column. The last column, labeled AVG, shows the average of the percentages of correct predictions in the preceding columns. The highest percentage(s) in each section of a column are marked with an asterisk. The highest percentage in the column is displayed in italics face.

was obtained by using the longest common subsequence technique, with 92% accuracy in distinguishing PDFs from BMPs, 90% accuracy in distinguishing PDFs from GIFs and 85% accuracy in distinguishing JPGs from GIFs (15% points lower than the 100% accuracy in Table 2). The linear discriminant technique worked best in distinguishing BMPs from JPGs and GIFs. We suspect that the inclusion of *CorNext* and *CorNextFreq* was useful in achieving a high rate of correct prediction. Since BMPs are not compressed, it is likely that the correlations between successive bytes in the file and the frequencies of successive byte values are higher for BMPs than for the other file types we used. The overall best accuracy in Table 3 was obtained from the longest common subsequence algorithm, with an average rate of correct prediction of 86%, slightly better than the 82.2% average rate of correct prediction when using ASCII frequency and entropy.

It should be noted that the longest common substring and longest common subsequence algorithms require the unknown fragment to be compared with each of the sample files. For each comparison, the runtime is of order mn, where m and n are the lengths of the unknown fragment and the sample file. Thus the total time complexity is of order mnt, where t is the number of sample files. Although we used a very fast algorithm for computing the longest common subsequence, it is much slower than the linear discriminant method. The classification model for the linear discriminant can be calculated quickly and in advance. To predict the type of the unknown fragment with this method our algorithm needs only to compute statistics based on frequencies and then use the statistics to evaluate a linear function. The time complexity is linear in the length of the fragment. Since we have obtained slightly better results in some cases using the longest common subsequence, it would be reasonable to use this technique when the reliability of the predictions is paramount, and the size and number of unknown file fragments are not too large. On the other hand, the linear discriminant technique would be a better choice when one must quickly identify the file types of large fragments or of a large number fragments.

As discussed in Section 1, the previous research most similar to ours is the work of Veenman (2007). That work also used the Fisher linear discriminant, but with the fileprint, entropy and a measure of Kolmogorov complexity as the data. It is difficult to compare the results of our experiments directly with the results from Veenman's paper. Veenman's experiments used file fragments of 11 types, where we used four types. Veenman designed algorithms to solve both two-class and multi-class file fragment classification problems. However, where we used two file types for our two-class classification problem, Veenman used one file type as one class and the other 10 file types as the other class. We plan to do future experiments that are more comparable to Veenman's. At present, an analysis of the data suggests that the results are similar for identification of JPG files using the linear discriminant. Our method appears to be more accurate in identifying BMP and PDF files, possibly due to our use of some particular statistics such as *Ascii* and *CorNext* that are particularly useful in identifying these file types. Furthermore, in many case the longest common subsequence algorithm appears to achieve greater accuracy than ours or Veenman's obtained using the Fisher linear discriminant. It should be noted, however, that the longest common subsequence algorithm is not as fast as the linear discriminant, as discussed above.

5. Other experiments

So far, we have implicitly considered a file fragment to be a block of contiguous bytes taken intact from the middle of a file. Fragments defined in this way are of natural interest in computer forensics. Here we briefly mention the result of applying our *LongerSSeqA* algorithm to fragments of a different kind: files corrupted by random errors. We compared JPG and PDF fragments exactly as in the previously described experiments, except that the fragments were obtained by corrupting each of the 1024 initial bytes of a file independently with 25% probability. By corrupting a byte, we mean changing it into another byte chosen uniformly at random. The algorithm correctly predicted the type of every fragment. The *LongerSSeqA* algorithm was nearly perfect.

Of course, no conclusions can be drawn from such a limited experiment, but the result agrees with the intuitive idea that substrings and subsequences of bytes from files of practical interest retain reliable type-identifying information even when significant fractions of the files have been corrupted by random errors. Similar experiments with a variety of error distributions may help to isolate and identify unique type-identifying information.

Along these lines, we worked with one of our students, David Reichert, to attempt to determine *particular* substrings common to typical files of a given type. More precisely, for each type under consideration we computed the longest substring that was common to at least 90% of the test files. (This is not feasible for subsequences: in general, the problem of computing longest common subsequences of arbitrarily many strings is NP-hard (Maier, 1978).) The computation involved the routine construction of a generalized suffix tree for the test files. The fragments' types were predicted by searching for embedded copies of the substrings computed for each type. As would be expected, the predictions were reasonably accurate when the header was included, but not accurate without the header. The method is therefore not useful for identifying the file types of fragments from the middle of a file, but it could be used to automatically find a string near the start of a file that can be used for type identification. This could be particularly useful when new proprietary file formats are introduced.

6. Directions for further development

We consider the software we have developed so far to be a proof-of-concept rather than a practical tool. We have several ideas for improving the accuracy and efficiency of our software. For convenience we have used two-group classifications in our experiments. For practical purposes, it would be necessary to classify file fragments into any of the many file types found on computers. Both the linear discriminant and the longest common subsequence technique can easily be extended to multi-group classifications. We have already performed successful experiments with four-group classifications using the longest common subsequence technique. In future work, we intend to investigate the behavior of both

the longest common subsequence and linear discriminant techniques when applied in a practical environment with a large number of potential file types.

The accuracy of the linear discriminant method could be improved by finding statistics and combinations of statistics that work better than the ones we have found so far. Statistics based on frequencies of two-byte (or longer) sequences might be quite useful for file type identification. The statistics we have used to this point are all based on frequencies of single bytes, but multi-byte codes are used in many file formats. Our search for good combinations of statistics has been essentially trial and error to this point. A more methodical search, possibly using a genetic algorithm, could lead to combinations of statistics that yield better performance.

There are also variations on the linear discriminant that might improve the performance of our algorithm. The quadratic discriminant yields a classification model composed of a set of quadratic functions. For some data, the quadratic discriminant works better than the linear discriminant. However, the quadratic discriminant may not work well in cases where the assumptions about normality and covariance of the data are not met. It remains to be seen whether the quadratic discriminant would be more effective than the linear discriminant for file type identification.

Another variation on the linear discriminant is to include a priori probabilities for the groups. In our work, that would mean assigning a priori probabilities for the different file types. Although in our artificial experiments, each file type was equally likely, the file types occur with differing probabilities in practice. By supplying the linear discriminant with these a priori probabilities, the overall rate of correct predictions could be increased.

A final minor modification of our linear discriminant program might help to decrease the number of situations in which the algorithm produces classification models that uselessly classify all of the unknown fragments into the same group, resulting in a 50% rate of correct prediction. The linear discriminant algorithm appears to fail in these cases either because the normality and covariance assumptions are not met, or due to numerical round-off errors in calculating a matrix inverse. We are using code for the linear discriminant obtained from the book *Classification Algorithms* by James (1985). The performance of this code might be improved by substituting matrix inversion code from a matrix software package such as JAMA.

We also may be able to achieve better performance using the greatest common subsequence algorithm. Two files of the same type in our test cases typically have longer subsequences in common than do files of different types, even when missing part of the header. We did not attempt to isolate the *parts* of the files that tended to contain these common subsequences, or to determine if the commonality is necessarily due to meta-data (e.g., files of one type may tend to have many instances of a data character that appears rarely in another). More refined experiments would be necessary for this. Along these lines, it may be worth computing the longest subsequence common to a large fraction of a *set* of files. As already mentioned, this is not feasible deterministically except for small sets, but genetic algorithms have been used for this task with some degree of success (Hinkemeyer and Julstrom, 2006).

Acknowledgments

We would like to thank David Reichert, a student at Bloomsburg University, for contributions to this project including acquiring files of various types for our tests, translation of computer code into Java, and his experiments with the longest common substring algorithm. We would also like to thank Stephen Rhein, also a student at Bloomsburg University, for designing a graphical user interface for our computer program. Finally, we are grateful to the anonymous referees for helpful comments and references to previous work in the area.

REFERENCES

Cormen Thomas H, Leiserson Charles E, Rivest Ronald L, Stein Clifford. Introduction to algorithms. 2nd ed. New York: McGraw-Hill; 2002.

Garfinkel SL. Carving contiguous and fragmented files with fast object validation. Digit Investig 2007;4S:S2–15.

Hinkemeyer Brena, Julstrom Bryant A. A genetic algorithm for the longest common subsequence problem. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. New York: ACM; 2006.

Huberty Carl J, Olejnik Stephen. Applied manova and discriminant analysis. 2nd ed. New York: Wiley; 2006.

James Mike. Classification algorithms. New York: Wiley; 1985.

JAMA: Java Matrix Package (Developed by Mathworks and National Institute of Standards). Available from: <http://math.nist.gov/javanumerics/jama/>.

Karresand Martin, Shahmehri Nahid. Oscar – file type identification of binary data in disk clusters and RAM pages. In: IFIP security and privacy in dynamic environments, vol. 201; 2006a. p. 413–424.

Karresand Martin, Shahmehri Nahid. File type identification of data fragments by their binary structure. In: Proceedings of the IEEE workshop on information assurance; 2006b. p. 140–7.

Li Wei-Jen, Wang Ke, Stolfo SJ, Herzog B. Fileprints: identifying file types by n-gram analysis. In: Proceeding of the 2005 IEEE workshop on information assurance; 2005. Available from: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1495935>.

Maier David. The complexity of some problems on subsequences and supersequences. J ACM 1978;25:322–36.

McDaniel Mason, Heydari M Hossain. Content based file type detection algorithms. In: Proceedings of the 36th Hawaii international conference on system sciences, Track 9. Washington, D.C.: IEEE Computer Society; 2003. p. 332.1.

Mitchell Melanie. An introduction to genetic algorithms. Boston: MIT Press; 1996.

Richard Golden G, Roussev Vassil. Scalpel: a frugal, high-performance file carver. In: Proceeding of the 2005 digital forensics research workshop, DFRWS; August 2005. Available from: <http://www.digitalforensicssolutions.com/Scalpel/>.

Roelofs Greg. PNG: the definitive guide. Cambridge, MA: O'Reilly; 1999. Available from: http://www.libpng.org/pub/png/book/.

Shannon CE. The mathematical theory of communication. Bell System Tech J 1948;27:379–423, 623–56.

Shannon MM. Forensic relative strength scoring: ASCII and entropy scoring. Int J Digit Evid 2004;2:1–19.

Veenman Cor J. Statistical disk cluster classification for file carving. In: IEEE third international symposium on information assurance and security; 2007.

William C. Calhoun is an associate professor and the assistant chair of the Department of Mathematics, Computer Science and Statistics at Bloomsburg University of Pennsylvania. He holds a Ph.D. in mathematics from the University of California at Berkeley. His research interests include computability theory, algorithmic randomness, combinatorics and digital forensics. He is the editor of the new on-line journal HEX: The Collegiate Journal of Computer Forensics.

Drue Coles earned a Ph.D. in Computer Science from Boston University before joining the Department of Mathematics, Computer Science and Statistics at Bloomsburg University in 2004. His research is in error-correcting codes.