Annual ADFSL Conference on Digital Forensics, Security and Law

2015
Proceedings

May 19th, 3:30 PM

A Survey of Software-based String Matching Algorithms for Forensic Analysis

Yi-Ching Liao
Norwegian Information Security Laboratory, Gjøvik University College, yi-ching.liao@hig.no

Scholarly Commons Citation

Liao, Yi-Ching, "A Survey of Software-based String Matching Algorithms for Forensic Analysis" (2015). *Annual ADFSL Conference on Digital Forensics, Security and Law*. 2.
https://commons.erau.edu/adfsl/2015/tuesday/2

A SURVEY OF SOFTWARE-BASED STRING MATCHING ALGORITHMS FOR FORENSIC ANALYSIS

Yi-Ching Liao
Norwegian Information Security Laboratory
Gjøvik University College, Norway
yi-ching.liao@hig.no

ABSTRACT

Employing a fast string matching algorithm is essential for minimizing the overhead of extracting structured files from a raw disk image. In this paper, we summarize the concept, implementation, and main features of ten software-based string matching algorithms, and evaluate their applicability for forensic analysis. We provide comparisons between the selected software-based string matching algorithms from the perspective of forensic analysis by conducting their performance evaluation for file carving. According to the experimental results, the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992) and the Karp-Rabin algorithm (Karp & Rabin, 1987) have the minimized search time for identifying the locations of specified headers and footers in the target disk.

Keywords: string matching algorithm, forensic analysis, file carving, Scalpel, data recovery

1. INTRODUCTION

File carving is the process of extracting structured files from a raw disk image without the knowledge of file-system metadata, which is an essential technique for digital forensics investigations and data recovery. There is no guarantee that metadata exists to provide the location of each file within a file system, and file headers can be anywhere in a raw disk image. Therefore, it is inevitable for file carving applications to search every byte of a raw disk image, at the physical level, to locate specific file headers and footers of interest to the investigation. To minimize the overhead of searching for file headers and footers, it is important to employ a fast string matching algorithm for reducing the search time (Richard III & Roussev, 2005). In this paper, we summarize the concept, implementation, and main features of several software-based string matching algorithms, and provide comparisons between them from the perspective of forensic analysis.

The rest of this paper is organized as follows. Section 2 describes the state of the art, and summarizes other research on the survey of string matching algorithms. Section 3 illustrates the importance of file carving, and introduces the implementation of one of the most popular open source file carving application, Scalpel (Richard III & Roussev, 2005). Section 4 summarizes the concept, implementation, and main features of ten software-based string matching algorithms. Section 5 presents the experimental results of comparisons between different string matching algorithms from the perspective of forensic analysis. Finally, section 6 concludes this paper and provides recommendations for future work.

2. RELATED WORK

Baeza-Yates (R. A. Baeza-Yates, 1989) surveys several important string matching algorithms, and presents empirical results of the execution time for searching 1,000 random patterns in random texts and an English text. The evaluated algorithms include the brute force algorithm, the Knuth-Morris-Pratt algorithm (Knuth, Morris, & Pratt, 1977), the Boyer-Moore algorithm (Boyer & Moore, 1977) and its variants, the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992), and the Karp-Rabin algorithm (Karp & Rabin, 1987). The empirical results show that the Horspool algorithm (Horspool, 1980), a simplification of the Boyer-Moore algorithm (Boyer & Moore, 1977), is the best known algorithm for almost all pattern lengths and alphabet sizes.

Navarro (Navarro, 2001) presents an overview of the state of the art in approximate string matching, which tolerates a limited number of errors during string matching. The most important application areas of approximate string matching include computational biology (e.g. DNA and protein sequences), signal processing (e.g. speech recognition), and text retrieval (e.g. correction of misspellings and information retrieval). Navarro states that information retrieval is among the most demanding areas of approximate string matching, because it is about extracting relevant information from a large text collection. Navarro also demonstrates empirical comparisons among the most efficient algorithms by running them on three kinds of texts: DNA, natural language, and speech.

Tuck et al. (Tuck, Sherwood, Calder, & Varghese, 2004) regard the string matching algorithm as the essential component of modem intrusion detection systems, since intrusion detection systems depend heavily on the content identified in the packets by string matching algorithms. In addition to modifying the Aho-Corasick algorithm (Aho & Corasick, 1975) to reduce the resource overhead, Tuck et al. also explain some core string matching algorithms, such as the SFKSearch algorithm utilized for low memory situations in Snort and the Wu-Manber algorithm (Wu & Manber, 1994). Even though the average case performance of the modified Wu-Manber algorithm (Wu & Manber, 1994) is among the best of all multi-pattern string matching algorithms, its worst case performance is not better than the brute force algorithm.

AbuHmed et al. (AbuHmed, Mohaisen, & Nyang, 2007) introduce a survey on the deep packet inspection algorithms and their usage for intrusion detection systems. They regard the string matching algorithm complexity as one of the challenges for deep packet inspection, since the resource-consuming pattern matching will significantly decrease the throughput of intrusion detection systems. In their opinions, the string matching algorithms suffer from two factors: the computation operations during comparisons and the number of patterns to be compared. AbuHmed et al. list the Knuth-Morris-Pratt algorithm (Knuth et al., 1977), the Boyer-Moore algorithm (Boyer & Moore, 1977), the Aho-Corasick algorithm (Aho & Corasick, 1975), the AC BM algorithm (Coit, Staniford, & McAlerney, 2001), the Wu-Manber algorithm (Wu & Manber, 1994), and the Commentz Walter algorithm (Commentz-Walter, 1979) as the most famous software-based string matching algorithms, and present a throughput comparison between existing intrusion detection systems with their algorithms and hardware implementations.

3. FILE CARVING

File carving is the process of recovering files without the file-system metadata describing the actual file system, which is vitally important for digital forensics investigations and data recovery. File carving is essential for digital forensics investigations, because it is able to provide human-readable information, instead of low level details, for forensic investigators (Richard III & Roussev, 2005). File carving is also a topic of great interest to an enterprise, because raw file recovery can minimize the impact of data loss when the file system of a disk is damaged (Pungila, 2012).

Scalpel (Richard III & Roussev, 2005) is one of the most popular open source file carving application that runs on Linux and Windows. To reassemble files from fragments, Scalpel first reads the entire disk image with a buffer of size 10 MB, and searches for the locations of file headers and footers. Since the configuration file “scalpel.conf” includes the known header and footer patterns of different file formats, forensic investigators can customize the configuration file to specify their target file formats. After the initial pass over the disk image, Scalpel matches each file header with an appropriate footer. The newest public release of Scalpel utilizes a modified Boyer-Moore algorithm (Boyer & Moore, 1977) as the default string matching algorithm. Since this paper is to investigate the applicability of the software-based string matching algorithms for forensic analysis, we concentrate on the first phase of Scalpel, in which the locations of specified headers and footers are identified in the target disk.

4. STRING MATCHING ALGORITHMS

Since there is no guarantee that file-system metadata exists to provide the location of each file within a file system, searching every byte of a raw disk image is unavoidable for file carving applications to identify the locations of structured files. Therefore, employing a fast string matching algorithm is indispensable for minimizing the overhead of file carving applications. The objective of string matching algorithms is to find one or more occurrences of pattern in a text through the sliding window mechanism. In this paper, we denote the pattern length as m, the text length as n, and the alphabet size of pattern and text as σ. We summarize the concept, implementation, and main features of ten software-based string matching algorithms as follows:

4.1 The Brute Force Algorithm

The brute force algorithm checks for the pattern by shifting the window by exactly one position with the time complexity O(m×n). The algorithm can perform the string matching in any order without a preprocessing phase. During the searching phase, it performs 2n text character comparisons (Aoe, 1994). The worst case scenario of the brute force algorithm is searching for repetitive text and pattern. Moreover, the brute force algorithm requires constant extra space to back up the text stream.

4.2 The Boyer-Moore Algorithm

The Boyer-Moore algorithm (Boyer & Moore, 1977) and the Knuth-Morris-Pratt algorithm (Knuth et al., 1977) are among the most widely used single pattern matching algorithms, in which each pattern is searched within a given text separately. The Boyer-Moore algorithm is considered as the most efficient string searching algorithm in both theory and practice, and it has become the standard for practical string searching. To improve the performance of searching, it performs the string matching from right to left, and it requires a preprocessing phase to determine the possibility of large shifts in the window with the time complexity O(m+σ). The pre-computed functions for shifts in the window are “good-suffix shift” and “bad-character shift”. During the searching phase, it performs with the time complexity O(m×n) and at most 3n character comparisons (Aoe, 1994). The best performance of the Boyer-Moore algorithm is O(n/m), which improves as the length of pattern m increases.

4.3 The Knuth-Morris-Pratt Algorithm

Knuth et al. (Knuth et al., 1977) present the Knuth-Morris-Pratt algorithm with the time complexity proportional to the sum of the lengths of pattern and text, O(m+n), which is independent of the alphabet size. The algorithm performs the string matching from left to right, and it needs a preprocessing phase to construct a partial-match table with the time complexity O(m). The table determines how many characters to slide the pattern when a mismatch occurs. During the searching phase, it performs at most 2n-1 character comparisons (Aoe, 1994). The Knuth-Morris-Pratt algorithm is a practical algorithm for on-line search, and it can be modified for searching multiple patterns in one single search.

4.4 The Karp-Rabin Algorithm

Since hashing is able to provide a simple method to avoid a quadratic number of character comparisons, Karp and Rabin (Karp & Rabin, 1987) propose an efficient randomized pattern matching algorithm that only checks if the window of text similar to the pattern through the hashing function. Therefore, the algorithm can examine the resemblance without checking whether the pattern occurs at each position of the text. The algorithm demands a preprocessing phase to compute hash values with the time complexity O(m), and it performs with the time complexity O(m×n) during the searching phase (Charras & Lecroq, 2004). The Karp-Rabin algorithm can be easily extended to find multiple patterns; however, the arithmetic operations can be slower than character comparisons.

4.5 The Horspool Algorithm

The Horspool algorithm (Horspool, 1980) is a simplified version of the Boyer-Moore algorithm (Boyer & Moore, 1977), which only utilizes the precomputed “bad-character shift” function for shifts in the window. Even though utilizing the “bad-character shift” is inefficient for small alphabets, it can be effective when the alphabet size is large enough compared to the pattern length. The Horspool algorithm requires a preprocessing phase with the time complexity O(m+σ), and it performs in any order with the time complexity O(m×n) during the searching phase (Charras & Lecroq, 2004). Baeza-Yates (R. A. Baeza-Yates, 1989) conducts a survey on several important string matching algorithms, and the empirical results show that the Horspool algorithm is the best known algorithm for almost all pattern lengths and alphabet sizes.

4.6 The Quick Search Algorithm

Similar to the Horspool algorithm (Horspool, 1980), the Quick Search algorithm (Sunday, 1990) is also a simplified version of the Boyer-Moore algorithm (Boyer & Moore, 1977), which only utilizes the precomputed “bad-character shift” function for shifts in the window. Likewise, the Quick Search algorithm needs a preprocessing phase with the time complexity O(m+σ), and it performs in any order with the time complexity O(m×n) during the searching phase. However, the Quick Search algorithm has a quadratic worst case time complexity in the searching phase.

4.7 The Shift-Or Algorithm

The main idea of the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992) is to represent the search state as a number, and each search attempt performs a small number of arithmetic and logical operations. By utilizing the bitwise techniques, the Shift-Or algorithm can be efficient if the pattern length is smaller than the memory-word size of the machine. The Shift-Or algorithm demands a preprocessing phase with the time complexity O(m+σ), and the time complexity of its searching phase is O(n), which is independent of the alphabet size and the pattern length (Charras & Lecroq, 2004).

4.8 The Smith Algorithm

Different from the Quick Search algorithm (Sunday, 1990) depending on the statistics of the language to determine the order of comparisons, the Smith algorithm (Smith, 1991) is able to perform the string matching language independently. It utilizes the precomputed “bad-character shift” function for shifts in the window from the Horspool algorithm (Horspool, 1980) and the Quick Search algorithm (Sunday, 1990). The Smith algorithm requires a preprocessing phase with the time complexity O(m+σ), and it performs with the time complexity O(m×n) during the searching phase (Charras & Lecroq, 2004). Since the Smith algorithm is a language-independent algorithm with competitive performance, it can perform the string matching efficiently without the knowledge of the text type.

4.9 The Raita Algorithm

Since neither the pattern nor the text is random in practice, Raita (Raita, 1992) proposes a new implementation that makes use of the dependencies between successive symbols. The Raita algorithm can perform 21 to 27 percent faster than the Horspool algorithm (Horspool, 1980) with all pattern lengths. After comparing the last character of the pattern with the rightmost character of the text, it compares the first and then the middle character before comparing the rest of characters. The Raita algorithm needs a preprocessing phase with the time complexity O(m+σ), and it performs with the time complexity O(m×n) during the searching phase (Charras & Lecroq, 2004).

4.10 The Berry-Ravindran Algorithm

Berry and Ravindran (Berry & Ravindran, 1999) introduce a new string matching algorithm that is more efficient than the existing algorithms through over 1,500,000 separate experiments. The Berry-Ravindran algorithm is a composite of the Quick Search algorithm (Sunday, 1990) and another variant of the Boyer-Moore algorithm (Boyer & Moore, 1977), the Zhu-Takaoka algorithms. It performs the window shifts by considering the “bad-character shift” for the two consecutive text

Table 1 Time Complexity of String Matching Algorithms

Algorithm	**Preprocessing phase**	**Searching phase**
The brute force algorithm	N/A	$O(m \times n)$
The Boyer-Moore algorithm (Knuth et al., 1977)	$O(m + \sigma)$	$O(m \times n)$
The Knuth-Morris-Pratt algorithm (Boyer & Moore, 1977)	$O(m)$	$O(m + n)$
The Karp-Rabin algorithm (Karp & Rabin, 1987)	$O(m)$	$O(m \times n)$
The Horspool algorithm (Horspool, 1980)	$O(m + \sigma)$	$O(m \times n)$
The Quick Search algorithm (Sunday, 1990)	$O(m + \sigma)$	$O(m \times n)$
The Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992)	$O(m + \sigma)$	$O(n)$
The Smith algorithm (Smith, 1991)	$O(m + \sigma)$	$O(m \times n)$
The Raita algorithm (Raita, 1992)	$O(m + \sigma)$	$O(m \times n)$
The Berry-Ravindran algorithm (Berry & Ravindran, 1999)	$O(m + \sigma^2)$	$O(m \times n)$

characters to the right of the window. The Berry-Ravindran algorithm demands a preprocessing phase with the time complexity O(m+σ²), and it performs with the time complexity O(m×n) during the searching phase (Charras & Lecroq, 2004).

Table 1 summarizes the time complexity, including the preprocessing and searching phases, of the string matching algorithms described in this section. However, the theoretical analysis can only show how the algorithm is likely to perform, instead of the actual performance. Therefore, it is necessary to conduct true experiments in order to evaluate the performance of algorithms in practice.

5. EVALUATION RESULTS

To provide comparisons between multiple string matching algorithms described in section 4 from the perspective of forensic analysis, we deploy an experimental testbed implemented with VMware Workstation and Ubuntu 12.04.3 based on the AMD64 architecture. The virtual machine utilizes a single CPU core with 1GB of memory. To evaluate the performance of each string matching algorithm, we utilize two test images for Scalpel 2.0 to extract various file formats. The first image ”11-carve-fat.dd” (Nick Mikus, 2005a) is a raw partition image of a 65 MB FAT32 file system, and the second image ”12-carve-ext2.dd” (Nick Mikus, 2005b) is a raw partition image of a 129.4 MB EXT2 file system. Since the file formats within the two images include doc, gif, jpg, mov, pdf, wav, and wmv, to specify the target file formats, we include 12 known header and footer patterns in the configuration file ”scalpel.conf”, which is shown in Table 2.

Since this paper aims to evaluate the applicability of the software-based string matching algorithms for forensic analysis, we concentrate on the performance of each algorithm during the first phase of Scalpel, in which the locations of specified headers and footers are identified in the target disk. In order to get more accurate results, we revert to the same snapshot when we evaluate each algorithm, and all evaluation results reported in this paper are the average from repeating the experiments for 30 times. Moreover, to find out the algorithm performance for different file formats, we separate each file format in the configuration file ”scalpel.conf”, which is shown in Table 2. Table 3 presents the experimental results of the search time and the number of files carved for different file formats between ten string matching algorithms for the image ”11-carvefat.dd”.

Table 2: Header and Footer Patterns in the “scalpel.conf” Configuration File

File extension	Header	Footer
doc1	\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00 \xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00	
doc2	\xd0\xcf\x11\xe0\xa1\xb1	
gif	\x47\x49\x46\x38\x39\x61	\x00\x00\x3b
jpg1	\xff\xd8\xff\xe0\x00\x10	\xff\xd9
jpg2	\xff\xd8\xff\xe1	\xff\xd9
mov1	????moov	
mov2	????mdat	
mov3	????free	
pdf1	\%PDF	\%EOF\x0d
pdf2	\%PDF	\%EOF\x0a
wav	RIFF????WAVE	
wmv	\x30\x26\xB2\x75\x8E\x66\xCF\x11 \xA6\xD9\x00\xAA\x00\x62\xCE\x6C	

*We distinguish the file extension with different headers and footers by adding numbers to the file extension.

Table 3: Search Time (in secs) and Number of Files Carved for Image “11-carve-fat.dd”

Algorithm	doc1	doc2	gif	jpg1	jpg2	mov1	mov2	mov3	pdf1	pdf2	wav	wmv
brute force	0.68 3	0.54 3	0.62 1	0.62 2	0.79 1	1.27 1	1.26 2	1.25 1	0.62 1	0.62 2	0.54 1	0.53 2
Boyer-Moore[1]	0.87 3	0.43 3	0.47 1	0.61 2	0.55 1	0.44 1	0.45 2	0.45 1	0.46 1	0.46 2	0.45 1	0.45 2
KMP	0.84 3	0.69 3	0.75 1	0.80 2	1.02 1	0.78 1	0.78 2	0.78 1	0.79 1	0.79 2	0.69 1	0.69 2
Karp-Rabin	0.20 3	0.17 3	0.18 1	0.19 2	0.24 1	0.17 0	0.17 0	0.17 0	0.19 1	0.19 2	0.17 0	0.17 2
Horspool	0.88 3	0.43 3	0.48 1	0.66 2	0.59 1	0.42 1	0.42 0	0.42 1	0.47 1	0.48 2	0.42 0	0.47 2
Quick Search	0.86 3	0.43 3	0.49 1	0.60 2	0.60 1	0.49 1	0.50 0	0.50 1	0.48 1	0.48 2	0.41 1	0.46 2
Shift-Or	0.16 3	0.13 3	0.14 1	0.15 2	0.18 1	0.13 0	0.13 0	0.13 0	0.15 1	0.15 2	0.13 0	0.13 2
Smith	1.04 3	0.52 3	0.57 1	0.72 2	0.72 1	0.58 1	0.58 0	0.58 1	0.57 1	0.58 2	0.50 1	0.55 2
Raita	0.60 3	0.30 3	0.34 1	0.53 2	0.46 1	0.29 0	0.29 0	0.29 0	0.34 1	0.34 2	0.28 0	0.33 2
Berry-Ravindran	0.90 3	0.45 3	0.49 1	0.51 2	0.62 1	0.52 1	0.52 1	0.51 1	0.51 1	0.51 2	0.43 0	0.42 2

[1] The modified Boyer-Moore algorithm that Scalpel utilizes

According to the experimental results from Table 3, some carved files are missed when utilizing the Karp-Rabin algorithm (Karp & Rabin, 1987), the Horspool algorithm (Horspool, 1980), the Quick Search algorithm (Sunday, 1990), the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992), the Smith algorithm (Smith, 1991), the Raita algorithm (Raita, 1992), and the Berry-Ravindran algorithm (Berry & Ravindran, 1999). The Karp-Rabin algorithm (Karp & Rabin, 1987), the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992), and the Raita algorithm (Raita, 1992) are unable to discover mov and wav file formats. The Horspool algorithm (Horspool, 1980), the Quick Search algorithm (Sunday, 1990), and the Smith algorithm (Smith, 1991) cannot locate the mov2 file format. In addition to mov2 file format, the Horspool algorithm (Horspool, 1980) also has problems finding wav file format. The Berry-Ravindran algorithm (Berry & Ravindran, 1999) is unable to discover wav file format either. However, it is able to locate one mov2 file. It appears that the types missed are those with the “?” character in the header pattern and with no footer pattern, which we regard as an open problem for future work.

Since there is no difference between the number of files carved by string matching algorithms for the image ”12-carve-ext2.dd” (3 doc1, 3 doc2, 1 gif, 3 jpg1, 1 pdf1, and 2 pdf2 files), Table 4 only shows the experimental results of the search time for different file formats between ten string matching algorithms for the image ”12-carve-ext2.dd”. Figure 1 and Figure 2 present the clear comparisons of search time for different file formats between different string matching algorithms for the images ”11-carve-fat.dd” and ”12-carve-ext2.dd” accordingly.

Table 4 Search Time (in secs) for Image “12-carve-fat.dd”

Algorithm	**doc1**	**doc2**	**gif**	**jpg1**	**jpg2**	**mov1**	**mov2**	**mov3**	**pdf1**	**pdf2**	**wav**	**wmv**
brute force	1.13	1.06	1.14	1.11	1.07	2.46	2.44	2.47	1.11	1.11	1.07	1.07
Boyer-Moore[1]	1.86	0.98	1.02	1.34	1.02	1.03	1.02	1.02	1.03	1.03	1.02	1.06
KMP	1.47	1.36	1.43	1.42	1.37	1.56	1.56	1.63	1.45	1.42	1.37	1.37
Karp-Rabin	0.35	0.33	0.34	0.34	0.33	0.33	0.33	0.33	0.34	0.34	0.33	0.33
Horspool	1.86	1.00	1.04	1.45	1.04	0.96	0.96	0.96	1.05	1.05	0.94	1.09
Quick Search	1.82	0.99	1.06	1.31	1.05	1.11	1.11	1.12	1.06	1.05	0.93	1.18
Shift-Or	0.26	0.25	0.26	0.26	0.25	0.25	0.25	0.25	0.26	0.26	0.26	0.25
Smith	2.18	1.20	1.26	1.57	1.25	1.31	1.33	1.33	1.28	1.27	1.15	1.30
Raita	1.21	0.68	0.72	1.14	0.72	0.66	0.66	0.66	0.74	0.73	0.64	0.78
Berry-Ravindran	1.89	1.03	1.07	1.05	1.07	1.13	1.13	1.13	1.08	1.08	0.97	0.95

[1] The modified Boyer-Moore algorithm that Scalpel utilizes

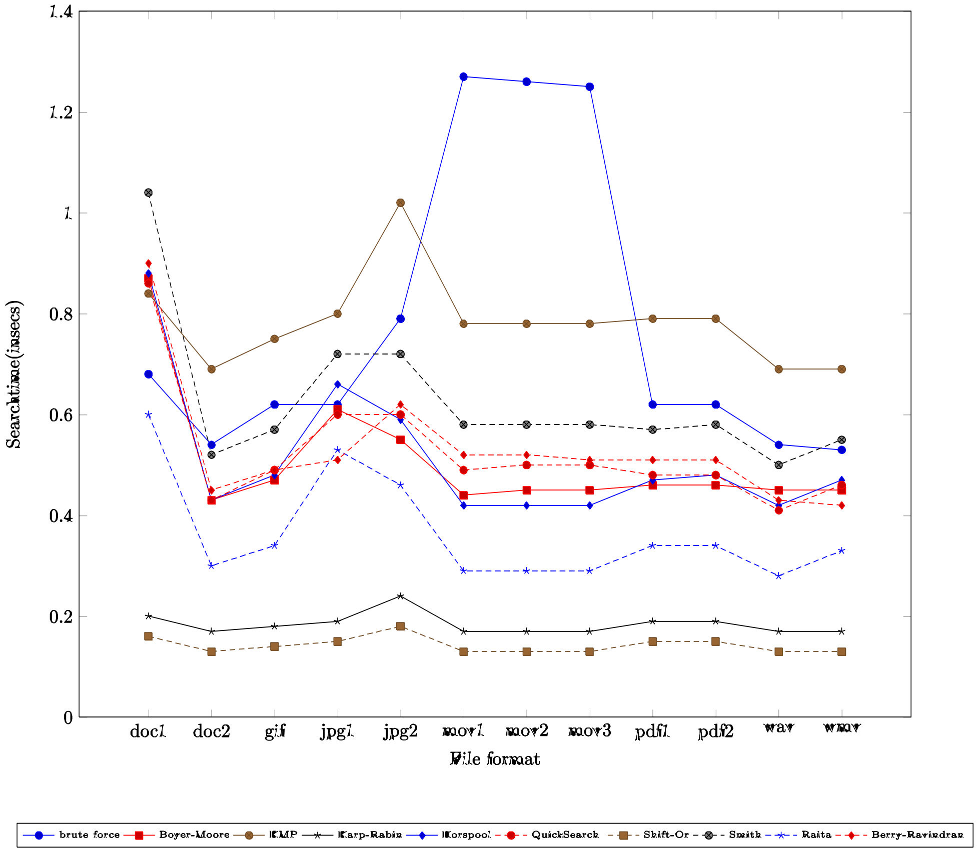

Figure 1 Search Time Comparison for Image ”11-carve-fat.dd”

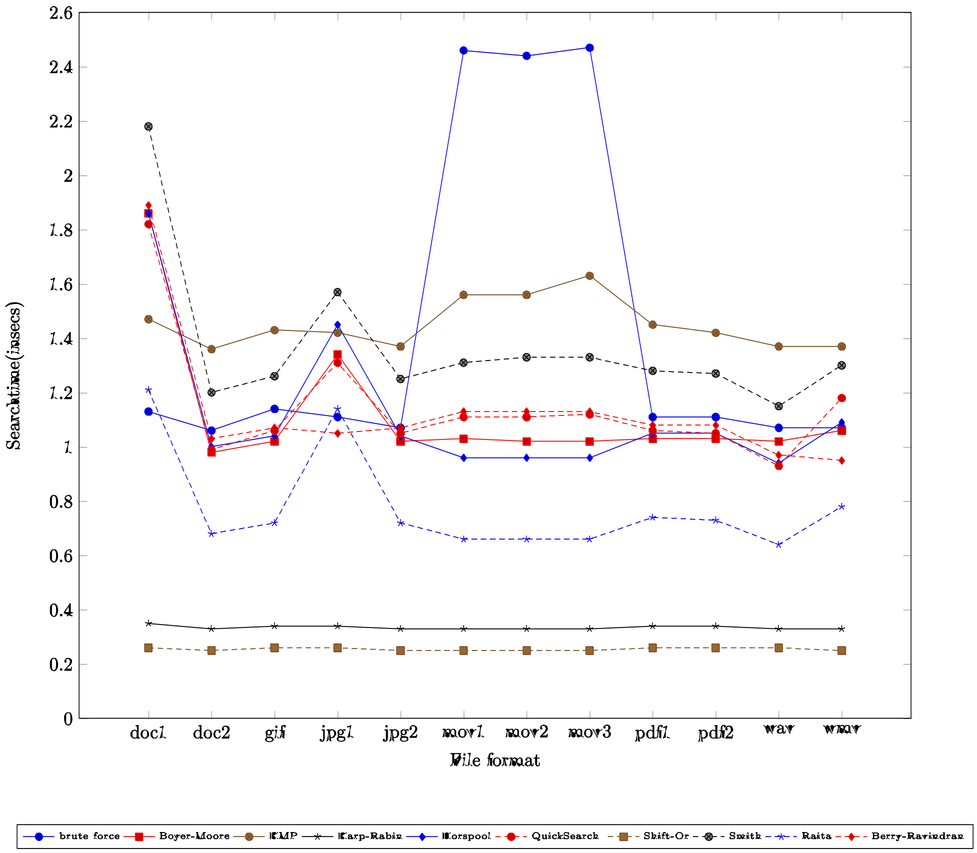

Figure 2 Search Time Comparison for Image ”12-carve-fat.dd”

According to Figure 1 and Figure 2, the experimental results show the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992) and the Karp-Rabin algorithm (Karp & Rabin, 1987) have the minimized execution time during the first phase of Scalpel, in which the locations of specified headers and footers are identified in the target disk. However, they both suffer from identifying the mov and wav file formats, which can be improved in the future.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we summarize the concept, implementation, and main features of ten software-based string matching algorithms, and provide comparisons between them from the perspective of forensic analysis. Since the theoretical analysis can only show how the algorithm is likely to perform, not the actual performance, we conduct true experiments to survey the performance of ten software-based string matching algorithms through utilizing them for file carving, which is an essential technique for digital forensics investigations and data recovery. Our experimental results show the Shift-Or algorithm (R. Baeza-Yates & Gonnet, 1992) and the Karp-Rabin algorithm (Karp & Rabin, 1987) have the minimized search time for identifying the locations of specified headers and footers in the target disk.

Even though file carving is an essential technique for digital forensics investigations and data recovery, there are other application areas in forensic analysis eager for better string matching algorithms, such as information retrieval and digital forensic text string searches. Moreover, there are several more string matching algorithms for future evaluation, including the AC BM algorithm (Coit et al., 2001), the Wu-Manber algorithm (Wu & Manber, 1994), the Commentz Walter algorithm (Commentz-Walter, 1979), and the Aho-Corasick algorithm (Aho & Corasick, 1975). Even though the evaluation method is valid, the evaluation results can be more unbiased if more test images are utilized. In addition to the execution time, other evaluation criteria, such as the storage overhead, CPU usage, and accuracy, can also be considered as future work.

ACKNOWLEDGMENT

Yi-Ching Liao is supported by the COINS Research School of Computer and Information Security.

REFERENCES

[1] AbuHmed, T., Mohaisen, A., & Nyang, D. (2007, November). A survey on deep packet inspection for intrusion detection systems. *Magazine of Korea Telecommunication Society, 24, 25–36*. arXiv:0803.0037

[2] Aho, A. V., & Corasick, M. J. (1975, June). Efficient string matching: an aid to bibliographic search. *Commun. ACM, 18(6), 333–340*. doi: 10.1145/360825360855

[3] Aoe, J.-i. (1994). Computer algorithms: string pattern matching strategies (Vol. 55). Wiley. com.

[4] Baeza-Yates, R., & Gonnet, G. H. (1992, October). A new approach to text searching. *Commun. ACM, 35(10), 74–82*. doi: 101145/135239.135243

[5] Baeza-Yates, R. A. (1989, April). Algorithms for string searching. *SIGIR Forum, 23(3-4), 34–58*. doi: 10.1145/74697.74700

[6] Berry, T., & Ravindran, S. (1999). A fast string matching algorithm and experimental results. In *Stringology (pp. 16–28)*.

[7] Boyer, R. S., & Moore, J. S. (1977, October). A fast string searching algorithm. *Commun. ACM, 20(10), 762–772*. doi: 10.1145/359842.359859

[8] Charras, C., & Lecroq, T. (2004). Handbook of exact string matching algorithms. King's College Publications.

[9] Coit, C., Staniford, S., & McAlerney, J. (2001). Towards faster string matching for intrusion detection or exceeding the speed of snort. In *DARPA information survivability conference amp; exposition II, 2001. DISCEX '01. proceedings (Vol. 1, pp. 367– 373 vol.1)*. doi: 10.1109/DISCEX.2001.932231

[10] Commentz-Walter, B. (1979). A string matching algorithm fast on the average. Automata, *Languages and Programming, 6th Colloquium, 71, 118–132.*

[11] Horspool, R. N. (1980). Practical fast searching in strings. *Software: Practice and Experience, 10(6), 501– 506*. doi: 10.1002/spe.4380100608

[12] Karp, R. M., & Rabin, M. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development, 31(2), 249–260*. doi: 10.1147/ rd.312.0249

[13] Knuth, D. E., Morris, J., & Pratt, V. R. (1977). Fast pattern matching in strings. *SIAM journal on computing, 6(2), 323–350*. doi:10.1137/0206024

[14] Navarro, G. (2001, March). A guided tour to approximate string matching. *ACM Comput. Surv., 33(1), 31–88*. doi: 10.1145/ 375360.375365

[15] Nick Mikus. (2005a, March). Digital forensics tool testing image, available at http://dftt.sourceforge.net/test11/index.html.

[16] Nick Mikus. (2005b, March). Digital forensics tool testing image, available at http://dftt.sourceforge.net/test12/index.html.

[17] Pungila, C. (2012). Improved file-carving through data-parallel pattern matching for data forensics. In *2012 7th IEEE international symposium on applied computational intelligence and informatics (SACI) (pp. 197–202)*. doi: 10.1109/SACI.2012.6250001

[18] Raita, T. (1992). Tuning the boyer-moore-horspool string searching algorithm. *Software: Practice and Experience, 22(10), 879–884*. doi: 10.1002/spe.4380221006

[19] Richard III, G. G., & Roussev, V. (2005). Scalpel: A frugal, high performance file carver. Refereed *Proceedings of the 5th Annual Digital Forensic Research Workshop*. Retrieved 2014-12-12, from http://www.dfrws.org/2005/proceedings/richard_scalpel.pdf

[20] Smith, P. D. (1991). Experiments with a very fast substring search algorithm. *Software: Practice and Experience, 21(10), 1065–1074*. doi: 10.1002/spe.4380211006

[21] Sunday, D. M. (1990, August). A very fast substring search algorithm. *Commun. ACM, 33(8), 132–142*. doi: 10.1145/79173.79184

[22] Tuck, N., Sherwood, T., Calder, B., & Varghese, G. (2004). Deterministic memory

efficient string matching algorithms for intrusion detection. In *INFOCOM 2004. twenty-third Annual Joint conference of the IEEE computer and communications societies (Vol. 4, pp. 2628–2639 vol.4)*. doi: 10.1109/INFCOM.2004.1354682

[23] Wu, S., & Manber, U. (1994). A fast algorithm for multi-pattern searching (Tech. Rep.). Technical Report TR-94-17, University of Arizona. Retrieved 2014-12-12, from http://webglimpse.net/pubs/TR94-17.pdf