Using parallel distributed processing to reduce the computational time of digital media similarity measures

Myeong Lim, James Jones

Chapter 4

USING PARALLEL DISTRIBUTED PROCESSING TO REDUCE THE COMPUTATIONAL TIME OF DIGITAL MEDIA SIMILARITY MEASURES

Myeong Lim and James Jones

Abstract Digital forensic practitioners are constantly challenged to find the best allocation of their limited resources. While automation will continue to partially mitigate this problem, the preliminary question about which media should be prioritized for examination is largely unsolved. Previous research has developed methods for assessing digital media similarity that may aid in prioritization decisions. Similarity measures may also be used to establish links between media and, by extension, the individuals or organizations associated with the media. However, similarity measures have high computational costs that delay the identification of digital media warranting immediate attention and render link establishment across large collections of data impractical.

This chapter presents and validates a method for parallelizing the computations of digital media similarity measures to reduce the time requirements. The proposed method partitions digital media and distributes the computations across multiple processors. It then combines the results as an overall similarity measure that preserves the accuracy of the original method executed on a single processor. Experiments on a limited dataset demonstrate reductions of up to 51% in processing time. The reductions vary based on the number of partitions chosen and specific digital media being examined, suggesting the need for additional testing and optimization strategies.

Keywords: Drive similarity, sector hashes, Jaccard index, parallel computation

1. Introduction

Digital forensic practitioners extract and process evidence from digital sources and media during the course of criminal and other investigations.

Digital evidence is fragile and volatile, requiring the attention of trained specialists to ensure that content of evidentiary value can be effectively isolated and extracted in a forensically-sound manner. This work is often time-sensitive and practitioners have limited time and resources, rendering early triage and prioritization of digital evidence a necessity.

As more digital data is created and digital storage systems grow in volume, digital forensic practitioners are overwhelmed by the massive amounts of data to be analyzed. Backlogs in digital forensic laboratories are common. According to a report by the FBI Regional Computer Forensics Laboratory Program [21], more than 15,000 digital devices and storage media were previewed and six petabytes of data were processed by the FBI in 2017 alone. Several Regional Computer Forensics Laboratories set the reduction of backlogs as an explicit goal.

Digital forensic practitioners seek to prioritize the data sources to be analyzed given limited resources and time. Manual and forensic-tool-based analyses may take many hours to complete for each data source. Even with automated tools such as EnCase [7], FTK and Autopsy, additional human review time is required before forensic analyses of drives can be conducted. Practitioners often do not have adequate information to make decisions about which media to work on first, something that can only be determined by spending valuable time and resources to evaluate each candidate source. The lack of efficient tools and knowledge about potential evidence in devices cause inefficiencies that can lead to critical deadlines being missed and delays in disseminating actionable information.

In the face of limited time, digital forensic practitioners must pick and choose which digital media to review from among many, rendering media triage a necessity. While triage tools exist for explicit tasks such as finding substrings of interest or specific files, a general-purpose triage method based on similarity measures between arbitrary-sized content and a labeled collection of digital media images is required. For example, a hard disk image that has high similarity to a cluster of previously-labeled drive images of interest can be prioritized for further analysis. Media similarity may also be used to infer relationships between entities, and as the basis for examining additional media and supporting the collection and analysis of additional evidence.

This chapter presents and validates a method for parallelizing the computation of a digital media similarity measure called the Jaccard index with normalized frequency (JINF) [14]. JINF, which is a variant of the Jaccard index, is based on digital media sector content comparisons. The computations adjust set intersection and union counts according to the frequencies of the set members, which are the contents of digital

media sectors in the application. The JINF similarity measure applies to digital media of varying size, operating systems, filesystems and form factors.

The proposed parallelization method splits the digital media of interest into N equal partitions. Each partition is distributed to a separate processor for computation of the precursor values for the JINF similarity measure. The precursor operation is the most computationally intensive part because it involves the processing of each sector in the media to be compared. The precursor values are then passed back to a central processing node, which aggregates the results and computes the final similarity measure. The final computation is a modified version of the original JINF computation that leverages the distributed precursor values while retaining the results and accuracy of the original JINF computation.

2. Previous Work

Several digital forensic algorithms and tools use string searches as their basis. The strings may be user-specified regular expressions that match features such as email addresses, telephone numbers, social security numbers, credit card numbers, network IP addresses and other information corresponding to pseudo-unique identifiers [9, 12, 19, 29]. Garfinkel [8] defines a pseudo-unique identifier as an identifier with sufficient entropy in a given corpus that it is highly unlikely to be repeated by chance.

Garfinkel [8] also noted that hard drive images are not regularly correlated with other images. He listed three problems: (i) improper prioritization, (ii) lost opportunities for data correlation and (iii) improper emphasis on document recovery. He attempted to address these problems via cross-drive analysis using pseudo-unique information such as social security numbers, credit card numbers and email addresses. In his approach, feature extractors analyzed the extracted string files and wrote their results to feature files. The extracted features were then applied to a multi-drive corpus to identify associations between drives. The research described in this chapter also attempts to address these three problems by providing digital forensic professionals with rapid media triage and prioritization capabilities, as well as a means for identifying previously-unknown associations between digital media and the entities using the devices containing the media without reliance on document recovery.

In the case of second-order cross-drive analysis, a different question is raised: Which drives in a corpus have the largest number of features in

common? To answer this question, Garfinkel [8] implemented the Multi Drive Correlator (MDC) that takes in a set of drive images with a feature to be correlated and outputs a list of (feature, drive-list) tuples. The program reads multiple feature files and generates a report that shows the number of drives in which each feature was seen, total number of times each feature was seen in the drives and list of drives in which each feature occurred.

Beverly et al. [1] extended this work using Ethernet media access control (MAC) addresses extracted from validated IP packets. They treated MAC addresses and drive images as nodes, and addresses on a hard drive image as links in a graph. They partitioned the graph to obtain distinct clusters in the collection of drive images.

Young et al. [31] introduced a file-agnostic approach that leverages hashing speed. Their approach employs sector hashes instead of file hashes. It compares blocks (fixed-sized file fragments) against a large dataset of sector hashes and considers individual sectors and collections of contiguous sectors (blocks or clusters). The approach is based on two hypotheses:

- If a block of data in a file is distinct, then a copy of the block found in a data storage device constitutes evidence that the file is or was present.
- If the blocks in a file are shown to be distinct with respect to a large and representative corpus, then the blocks can be treated as if they are universally distinct.

Young et al. [31] suggest that analyses of digital media would be more accurate and faster if a database of hash values computed from fixed-sized blocks of data is used. They employed large corpora such as Govdocs [10] and NSRL RDS [17] to populate a hash value database. Three types of sectors – singleton, paired and common sectors – were analyzed to understand the root causes of non-distinct blocks. They discovered that common sectors were typically encountered when the same blocks were present in multiple files due to malware code reuse and common file container formats.

In order to implement a field deployment on a laptop, Young and colleagues considered sampling sectors instead of processing all the media sectors. Several database implementations were considered and a Bloom filter front-end was ultimately implemented to speed up generic query times [3]. Young and colleagues analyzed several filesystems to demonstrate the generality of their approach. However, encrypted files and filesystems were found to be problematic because the (same) data of interest is stored differently when encrypted.

Moia et al. [15] assessed the impact of common blocks on similarity. They showed how common data can be identified and how the data is spread over various file types and their frequencies. They observed that common data is often generated by applications, not by users. They also demonstrated that removing common data reduced the number of matches by approximately 87%.

Garfinkel and McCarrin [11] have proposed hashing blocks instead of entire files. This block hashing method inspired the drive similarity measure methodology proposed in [14]. Garfinkel and McCarrin also specified the HASH-SETS algorithm that identifies the existence of files and the HASH-RUN algorithm that reassembles files using a database of file block hashes. A fixed block size (e.g., 4 KiB) may present a problem due to filesystem alignment. However, this is addressed by hashing overlapping blocks with a 4 KiB sliding window over the entire drive and moving the window one sector at a time.

Taguchi [27] experimented with different sample sizes using random sampling and sector hashing for drive triage. Given a drive, the goal was to provide a digital forensic practitioner with information about the utility of continuing an investigation. If a block hash value of target data is in the database, then it is very likely that the target file is on the drive. However, if no hashes are found during sampling, then a confidence level is computed to express the likelihood that the target data is not on the drive.

The `spamsum` program developed by Tridgell [28] performs context-triggered piecewise hashing to find updates of files. It identifies email messages that are similar to known spam. The `ssdeep` program [13], based on `spamsum`, computes and matches context-triggered piecewise hash values. It is more effective than `spamsum` for relatively small objects that are similar in size. However, it is vulnerable to attacks that insert trigger sequences at the beginning of files, exploiting the fact that an `ssdeep` signature value can have at most 64 characters [4].

Roussev et al. [23–25] have developed a similarity digest hashing method implemented in a program called `sdhash`. The `sdhash` program finds the features in a neighborhood with the lowest probability of being encountered by chance. Each selected feature, which is a 64-byte sequence, is hashed and placed in a Bloom filter. When the Bloom filter reaches full capacity, a new filter is generated. Thus, a similarity digest is a collection of a sequence of Bloom filters.

Breitinger et al. [5] developed MRSH-v2, which is based on the MRS hash [26] and context-triggered piecewise hashing. The algorithm uses a sequence of Bloom filters for fast comparison instead of a Base64-encoded fingerprint. It divides an input into chunks using a rolling hash. Each

chunk is hashed and inserted into a Bloom filter. Like `sdhash`, MRSH-v2 has a variable length fingerprint, targeting 0.5% of the input length. Specific inputs determine whether the algorithm runs in the fragment detection and file similarity modes.

Oliver et al. [18] have proposed a locality-sensitive hashing methodology called TLSH. TLSH populates an array of bucket counts by processing an input byte sequence using a sliding window. Quartile points are computed from the array, following which digest headers and bodies are constructed. The digest header values are based on the quartile points, file length and checksum. A digest body comprises a sequence of bit pairs determined by each bucket value in relation to the quartile points. A distance score is assigned between two digests; this score is the summed-up distance between the digest headers and bodies. The distance between two digest headers is based on file lengths and quartile ratios. The distance between two digest bodies is computed as the Hamming distance. Experiments indicate that TLSH is more robust to random adversarial manipulations than `ssdeep` and `sdhash`.

Penrose et al. [20] used a Bloom filter for rapid contraband file detection. The Bloom filter reduces the size of the hash database by an order of magnitude, but incurs a small false positive rate. Penrose and colleagues subsequently implemented a larger Bloom filter for faster access, achieving 99% accuracy while scanning for contraband files in a test dataset within minutes.

Bjelland et al. [2] present three common scenarios where approximate matching can be applied: (i) search, (ii) streaming and (iii) clustering. In a search scenario, the data space is large compared with a streaming scenario. In a clustering scenario, the input and data spaces are the same. Approximate matching is impractical for large datasets due to its high latency.

Breitinger et al. [6] focus on approximate matching (i.e., similarity hashing or fuzzy hashing). They divide approximate matching methods into three main categories: (i) bytewise, (ii) syntactic and (iii) semantic matching. Bytewise matching relies only on the sequences of bytes that make up a digital object, without reference to any structures in the data stream, or to any meaning the byte stream may have when interpreted. The method described in this chapter can viewed as employing bytewise matching because it does not rely on the internal structure of a hard drive and does not give any meaning to the byte stream.

Moia and Henriques [16] have presented steps for developing new approximate matching functions. Approximate matching functions address the limitations of cryptographic hash functions that cannot detect non-identical, but similar, data.

The main goal of this research is to reduce the time required to compute JINF digital media similarity measures. Most of the methods described above can be parallelized in a manner similar to the proposed approach. As in the case of the JINF parallelization described in this chapter, most of the methods described above would also require certain modifications to be parallelized.

3. Jaccard Indexes of Similarity

This section describes the basic Jaccard index of similarity and the Jaccard index of similarity with normalized frequency.

3.1 Jaccard Index

The Jaccard index (JI) is a simple and widely-used similarity measure for arbitrary sets of data [22]. It is defined as the size of the intersection divided by the size of the union of sets A and B:

$$\mathrm{JI}(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad 0 \leq \mathrm{JI}(A,B) \leq 1$$

The weighted Jaccard index is used to express the similarity between two hard drives. Specifically, if $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ are two vectors with real values $x_i, y_i \geq 0$, then the weighted Jaccard index JI_w is defined as:

$$\mathrm{JI}_w = \frac{\sum_{i=1}^{n} \mathrm{Min}(x_i, y_i)}{\sum_{i=1}^{n} \mathrm{Max}(x_i, y_i)}$$

3.2 Jaccard Index with Normalized Frequency

Lim and Jones [14] introduced the basic Jaccard index with frequency (JIWF) and the Jaccard index with normalized frequency (JINF). JIWF is based on the Jaccard index and sector hashing. JINF is designed to address the sensitivity of JIWF to images of different sizes, for example, when comparing a thumb drive against a multi-terabyte hard drive.

JINF is described in detail because it is modified to parallelize the computations without affecting the numerical results. The new method, which is called the Jaccard index with split files (JISPLIT), is based on JINF with the addition of an intelligent splitting and modified computation mechanism.

In this work, the digital media of known interest is called the source drive and the media of investigatory interest is called the target drive. Parallelized computations are performed using the ARGO distributed computing platform.

The standard Jaccard index computation employs intersection and union. The computation of the new Jaccard index JINF, which is based on the weighted Jaccard index, employs modified definitions of $Intersection^*$ (I^*) and $Union^*$ (U^*):

$$Intersection^*(N1, N2) = \text{Min}(|N1|, |N2|)$$

$$Union^*(N1, N2) = \text{Max}(|N1|, |N2|)$$

where $N1$ and $N2$ are normalized frequencies.

The normalized frequency N_f is given by:

$$N_f = F_i / S_T$$

where F_i is the frequency of a sector hash value i and S_T is the total number of sectors in a drive.

JINF requires two normalized values to be computed for each distinct hash value, one for the source and the other for the target. As described in [14], non-probative sectors are removed before the JINF computation to improve similarity measure performance. The non-probative sectors, which are collected in a "whitelist," comprise NULL byte and SPACE byte sectors as well as sectors that appear in a clean operating system installation. The sectors written during operating system installation are not produced by user activity and, therefore, would not contribute to the similarity measure. The sector hashes computed for a clean operating system installation on test drives are saved in a database for pre-filtering (exclusion) purposes.

In general, the JINF similarity value is computed as:

$$\text{JINF}(S, T) = \frac{\text{Sum of } Intersection^*(S, T)}{\text{Sum of } Union^*(S, T)}$$

where S and T are the source and target drives, respectively.

Table 1 shows the hash values, sector frequencies and normalized sector frequencies for hypothetical source and target drives.

Table 2 shows the $Intersection^*$ and $Union^*$ values computed for the hypothetical source and target drives using the normalized frequency values in Table 1. The sum of $Intersection^*$ values over all the hashes is 0.6 and the sum of all $Union^*$ values is 1.4. The resulting JINF value is $0.6/1.4 = 0.4286$. Note that the JINF value is not dependent on which drive is the source and which drive is the target. The $Intersection^*$ and $Union^*$ are computed using the Min and Max of the normalized frequency of each sector hash from both drives.

Table 3 shows how the JINF values change when the frequency of sector hash A is successively increased by one in target drives $T2$, $T3$

Table 1. Hash values and frequencies of source drive S and target drive T.

Source Drive *S*			**Target Drive *T***		
Hash Value	**Frequency**	**Normalized Frequency**	**Hash Value**	**Frequency**	**Normalized Frequency**
A	5	0.3333	A	1	0.0667
B	4	0.2667	B	2	0.1333
C	3	0.2000	C	3	0.2000
D	2	0.1333	D	4	0.2667
E	1	0.0667	E	5	0.3333
Total	15	1	Total	15	1

Table 2. $Intersection^*$ and $Union^*$ of two normalized frequency values.

Hash Value	**Normalized Source Frequency**	**Normalized Target Frequency**	***Intersection** (I^*)**	***Union** (U^*)**
A	0.3333	0.0667	0.0667	0.3333
B	0.2667	0.1333	0.1333	0.2667
C	0.2000	0.2000	0.2000	0.2000
D	0.1333	0.2667	0.1333	0.2667
E	0.0667	0.3333	0.0667	0.3333
Total			0.6	1.4
JINF (S, T)			0.4286	

and $T4$ (normalized frequencies of the drives are not shown). As the frequency of the first block A in the target drive moves toward the frequency of the same sector hash A in the source drive S, the similarity should increase. Each drive in Table 3 is essentially a new target drive that is checked against the source drive S. For each target drive, the JINF similarity value increases when the frequency of sector hash A increases. Note that the total number of blocks increases by one as the frequency of sector hash A is increased by one. The increase in the total number of blocks reduces the similarity because the portion of each block against the total number of blocks decreases. In contrast, the positive effect of increasing the frequency of sector hash A is greater than the negative effect of increasing the total number of blocks.

Table 4 shows how the similarity values increase when the frequency of sector hash E is incorporated in the computations. $T6$ is a new target

Table 3. JINF values of target drives $T2$, $T3$ and $T4$.

Hash	***T2***			***T3***			***T4***		
	Freq.	I^*	U^*	**Freq.**	I^*	U^*	**Freq.**	I^*	U^*
A	**2**	0.1250	0.3333	**3**	0.1764	0.3333	**4**	0.2222	0.3333
B	2	0.1250	0.2667	2	0.1176	0.2667	2	0.1111	0.2667
C	3	0.1875	0.2000	3	0.1764	0.2000	3	0.1667	0.2000
D	4	0.1333	0.2500	4	0.1333	0.2352	4	0.1333	0.2222
E	5	0.0667	0.3125	5	0.0666	0.2941	5	0.0667	0.2778
Sum	16	0.6375	1.3625	17	0.6705	1.3294	18	0.7	1.3
JINF	$(S, T2) = 0.4678$			$(S, T3) = 0.5044$			$(S, T4) = 0.5384$		

Table 4. JINF values of target drives $T5$, $T6$ and $T7$.

Hash	***T5***			***T6***			***T7***		
	Freq.	I^*	U^*	**Freq.**	I^*	U^*	**Freq.**	I^*	U^*
A	5	0.2631	0.3333	5	0.2778	0.3333	10	0.0667	0.3333
B	2	0.1052	0.2667	2	0.1111	0.2667	20	0.1333	0.2667
C	3	0.1578	0.2000	3	0.1667	0.2000	30	0.2000	0.2000
D	4	0.1333	0.2106	4	0.1333	0.2222	40	0.1333	0.2667
E	**5**	0.0667	0.2631	**4**	0.0667	0.2222	50	0.0667	0.3333
Sum	19	0.7263	1.2736	18	0.7556	1.2444	150	0.6	1.4
JINF	$(S, T5) = 0.5702$			$(S, T6) = 0.6071$			$(S, T7) = 0.4286$		

drive created from target drive $T5$, where the frequency of sector hash E in target drive $T6$ is reduced by one (from five to four) in target drive $T5$. Target drive $T6$ has a JINF value of 0.6071, which is higher than the JINF value of 0.5702 of target drive $T5$. This is because the total number of blocks in target drive $T6$ is closer to the number in the source drive S and has less negative impact on the JINF value computation compared with target drive $T5$.

Target drive $T7$ in Table 4 demonstrates how well the method copes with a size difference between the target and source drives. The frequency of each block is copied from target drive T in the right-hand side of Table 1 and multiplied by ten. The JINF(S, T) and JINF(S, $T7$) values are the same because the normalized frequency of each hash block is the same for both drives. Therefore, JINF does not require the sizes of the drives to be the same, or even to be measured.

Woods et al. [30] have created a realistic forensic corpus, M57-Patents, that supports cyber security and digital forensics research. The multimodal corpus includes memory and hard drive images, network packet captures and images from USB drives and cellphones.

A scripted scenario was prepared when creating the corpus. Data in the corpus is simpler than real-world data, but it is complex enough to be useful for digital forensics research. The M57-Patents corpus [10] comprises 68 hard disk images that were taken from four computer systems named Pat, Terry, Jo and Charlie over a 25-day period; each system was imaged 17 times during the 25-day experiment. Lim and Jones [14] leveraged the M57 Patents corpus to validate the JINF similarity measure. A total of 305 hard disk images (target) were compared with four disk images (source). Except for the first few images, correct results were obtained with accuracy greater than 98%. Poor results for the first few images were likely due to limited user data at the start of the M57-Patents scenario, resulting in user images with minimal differences.

4. Jaccard Index with Split Files

This section discusses the Jaccard index with split files (JISPLIT). The *Target* image is split into a number of smaller files and similarity measures are computed between the *Source* image and smaller *Target* files. Asssume that the *Target* is split into ten small files: $Trgt_{sp_1}$, $Trgt_{sp_2}$, ..., $Trgt_{sp_{10}}$.

A pool-like class in the `mpi4py` library (like the one in the Python multiprocessing library) is employed. The `MPIPoolExecutor.map()` function handles the complexity of coordinating communications with nodes, farms the tasks and collects the results. To implement parallel processing, each computation of JINF(*Source*, $Trgt_{sp_i}$) is assigned to a node in the ARGO distributed computing platform.

Table 5 shows the results of the preliminary experiments, which establish that the sum of JINF values using the split files does not match the JINF value for two complete images.

To understand the difference, consider the *Source* and *Target* images shown in Tables 6 and 7.

The *Target* image is split into two files $T1$ and $T2$ shown in Table 8. Table 9 shows the hash frequencies of the two split files.

Tables 10 and 11 show the JINF(*Source*, $T1$) and JINF(*Source*, $T2$) computations, respectively. The sum of the two JINF values is 0.22222 + 0.15 = 0.37222. However, Table 5 shows that JINF(*Source*, *Target*) has a value of 0.3043.

Table 5. JINF(*Source*, *Target*).

Normalized Hash Count	I^*	U^*
A: (1/15, 2/15)	1/15	2/15
B: (1/15, 1/15)	1/15	1/15
C: (1/15, 1/15)	1/15	1/15
D: (1/15, 1/15)	1/15	1/15
E: (1/15, 1/15)	1/15	1/15
F: (1/15, 1/15)	1/15	1/15
G: (1/15, 1/15)	1/15	1/15
H: (1/15, 0/15)	0	1/15
I: (2/15, 0/15)	0	2/15
J: (2/15, 0/15)	0	2/15
K: (3/15, 0/15)	0	3/15
Q: (0, 2/15)	0	2/15
Y: (0, 2/15)	0	2/15
Z: (0, 3/15)	0	3/15
Sum	7/15	23/15
JINF(*Source*, *Target*) = $\frac{(7/15)}{(23/15)}$	=	**0.3043**

The discrepancy occurs because hash Q and hash Z are split into two different files. When a hash is split into multiple target files, the *Union** of the hash cannot have the same maximum value that it does in the non-split JINF computation. If the sum of the *Union** values is computed over all (*Source*, $Trgt_{sp_i}$) pairs, the *Union** value of the split hash is always less than the maximum *Union** value for (*Source*, *Target*). However, if the sum of the *Intersection** values is computed over all (*Source*, $Trgt_{sp_i}$) pairs, the result is the same as the sum of the *Intersection** values for (*Source*, *Target*).

To address this anomaly, pre-processing and post-processing must be performed before and after the ARGO platform is employed for distributed computations. The pre-processing step applies the `panda groupby` operation to the hash field, which sorts the hash values. This ensures that the same hash is placed in the same file during a split. In the example above, the hash values Q and Z had to be either in $T1$ or $T2$, both not in both files.

After the ARGO platform completes its parallel processing for each (*Source*, $Trgt_{sp_i}$) pair, each ARGO node returns the following two-tuple:

$$(value_i, \{(h1, NF(h1)), (h2, NF(h2)), \ldots, (hk, NF(hk))\}_i)$$

Table 6. *Source* layout.

Sector	Hash
1	K
2	B
3	C
4	I
5	J
6	K
7	G
8	H
9	I
10	D
11	J
12	E
13	A
14	K
15	F

Table 7. *Target* layout.

Sector	Hash
1	A
2	Z
3	Q
4	A
5	B
6	C
7	D
8	E
9	F
10	G
11	Y
12	Q
13	Y
14	Z
15	Z

Table 8. Layouts of the split files $T1$ and $T2$.

T1		***T2***	
Sector	**Hash**	**Sector**	**Hash**
1	A	1	E
2	Z	2	F
3	Q	3	G
4	A	4	Y
5	B	5	Q
6	C	6	Y
7	D	7	Z
		8	Z

where $NF(h)$ is the normalized frequency of a hash h and k is the total number of unique hashes assigned to each ARGO node i.

The first element of a tuple is the sum of the *Intersection** values (Table 10 or 11) and the second is the *Union** column with the corresponding hash values (Table 10 or 11). A separate ARGO node is assigned to perform the computations of JINF(*Source*, $Trgt_{sp_i}$) for each pair. Note that an ARGO node does not complete the normal JINF computations. Instead, it returns an intermediate result as a two-tuple:

Table 9. Hash frequencies of the split files $T1$ and $T2$.

T1		***T2***	
Hash	**Frequency**	**Hash**	**Frequency**
A	2	E	1
B	1	F	1
C	1	G	1
D	1	Q	1
Q	1	Y	2
Z	1	Z	2

Table 10. JINF(*Source*, *T1*).

Normalized Hash Count	I^*	U^*
A: (1/15, 2/15)	1/15	2/15
B: (1/15, 1/15)	1/15	1/15
C: (1/15, 1/15)	1/15	1/15
D: (1/15, 1/15)	1/15	1/15
E: (1/15, 0)	0	1/15
F: (1/15, 0)	0	1/15
G: (1/15, 0)	0	1/15
H: (1/15, 0)	0	1/15
I: (2/15, 0)	0	2/15
J: (2/15, 0)	0	2/15
K: (3/15, 0)	0	3/15
Q: (0, 1/15)	0	**1/15**
Y: (0, 0)	0	0
Z: (0, 1/15)	0	**1/15**
Sum	4/15	18/15
JINF(*Source*, *T1*) = $\frac{(4/15)}{(18/15)}$ = **0.2222**		

Table 11. JINF(*Source*, *T2*).

Normalized Hash Count	I^*	U^*
A: (1/15, 0)	0	1/15
B: (1/15, 0)	0	1/15
C: (1/15, 0)	0	1/15
D: (1/15, 0)	0	1/15
E: (1/15, 1/15)	1/15	1/15
F: (1/15, 1/15)	1/15	1/15
G: (1/15, 1/15)	1/15	1/15
H: (1/15, 0)	0	1/15
I: (2/15, 0)	0	2/15
J: (2/15, 0)	0	2/15
K: (3/15, 0)	0	3/15
Q: (0, 1/15)	0	**1/15**
Y: (0, 2/15)	0	2/15
Z: (0, 2/15)	0	**2/15**
Sum	3/15	20/15
JINF(*Source*, *T2*) = $\frac{(3/15)}{(20/15)}$ = **0.15**		

$$\text{JISPLIT}(Source, Target) = \frac{\text{Final sum of } Intersection^*}{\text{Final sum of } Union^*}$$

where

$$\text{Final sum of } Intersection^* = \sum_{i=1}^{N} \text{Sum of } Intersection^* \text{ from } node_i$$

and

$$\text{Final sum of } Union^* = \sum_{j=1}^{H} Union_j^* \text{ row in final } Union^* \text{ column}$$

Table 12. Hash frequencies of the split files from *Target*.

T_a		T_b	
Hash	**Frequency**	**Hash**	**Frequency**
A	2	G	1
B	1	Q	2
C	1	Y	2
D	1	Z	3
E	1		
F	1		

where N is the number of split files and H is the number of unique hashes.

The sum of *Intersection** values is the numerator in the final JISPLIT(*Source*, *Target*) computation, which is straightforward. In the case of the denominator in the final JINF computation, the central node is used to construct the final *Union** column using the *Union** columns returned the ARGO nodes.

The normalized frequency of each hash value h in the final *Union** column is $\text{Max}(NF(h)_1, NF(h)_2, \ldots, NF(h)_H)$, where $NF(h)_i$ is the normalized frequency of hash h from ARGO node i. The sum of all the rows in the final *Union** column is the denominator in the final JISPLIT(*Source*, *Target*) computation.

An example is helpful to illustrate the computations. The *Target* image is split into two files, T_a and T_b, by applying the `groupby` operation on the hash field of the *Target*. Table 12 shows the hash frequencies of the split files from *Target*.

Assume that ARGO $node_a$ computes JINF(*Source*, T_a) and ARGO $node_b$ computes JINF(*Source*, T_b). The $node_a$ returns 6/15 as the sum of *Intersection** along with the *Union** column, which is the last column in Table 13. The $node_b$ returns 1/15 as the sum of *Intersection** along with the *Union** column, which is the last column in Table 14.

The two-tuple returned by $node_a$ is (6/15, {(A, 2/15), (B, 1/15), (C, 1/15), ..., (K, 3/15)}). As shown in Table 15, the final sum of *Intersection** obtained via post-processing is computed as the sum of (6/15, 1/15), which is 7/15.

As shown in Table 16, the final *Union** column is constructed by selecting the maximum of the two columns returned by the two ARGO nodes. The denominator value is the sum of the final *Union** column, which is 23/15. The resulting JISPLIT value is $\frac{7/15}{23/15} = 0.3043$, which matches the JINF(*Source*, *Target*) value in Table 5.

Table 13. JINF(*Source*, T_a).

Normalized Hash Count	I^*	U^*
A: (1/15, 2/15)	1/15	**2/15**
B: (1/15, 1/15)	1/15	**1/15**
C: (1/15, 1/15)	1/15	**1/15**
D: (1/15, 1/15)	1/15	**1/15**
E: (1/15, 1/15)	1/15	**1/15**
F: (1/15, 1/15)	1/15	**1/15**
G: (1/15, 0)	0	**1/15**
H: (1/15, 0)	0	**1/15**
I: (2/15, 0)	0	**2/15**
J: (2/15, 0)	0	**2/15**
K: (3/15, 0)	0	**3/15**
Q: (0, 0)	0	**0**
Y: (0, 0)	0	**0**
Z: (0, 0)	0	**0**
Sum	**6/15**	

Table 14. JINF(*Source*, T_b).

Normalized Hash Count	I^*	U^*
A: (1/15, 0)	0	**1/15**
B: (1/15, 0)	0	**1/15**
C: (1/15, 0)	0	**1/15**
D: (1/15, 0)	0	**1/15**
E: (1/15, 0)	0	**1/15**
F: (1/15, 0)	0	**1/15**
G: (1/15, 1/15)	1/15	**1/15**
H: (1/15, 0)	0	**1/15**
I: (2/15, 0)	0	**2/15**
J: (2/15, 0)	0	**2/15**
K: (3/15, 0)	0	**3/15**
Q: (0, 2/15)	0	**2/15**
Y: (0, 2/15)	0	**2/15**
Z: (0, 3/15)	0	**3/15**
Sum	**1/15**	

Table 15. Final *Intersection** from two ARGO nodes.

Pair of Split Files	I^*
(*Source*, T_a)	6/15
(*Source*, T_b)	1/15
Final Sum	**7/15**

At this point, only the *Target* image is split. The *Source* image may also be split as shown in Table 17. The same pre-processing step is applied.

Four ARGO nodes are required because there are four pairs of split file combinations: (S_a, T_a), (S_a, T_b), (S_b, T_a) and (S_b, T_b). Tables 18 through 21 show the JINF computations for the four pairs.

Table 22 shows the *Intersection** values returned by the four ARGO nodes. The final sum of 7/15 is the numerator in the JISPLIT(*Source*, *Target*) computation.

Table 23 shows the final *Union** columns returned by the four ARGO nodes. The final sum of 23/15 is the denominator in the JISPLIT(*Source*, *Target*) computation. The JISPLIT value obtained from the four ARGO

Table 16. Final *Union** column.

Hash	$\boldsymbol{U^*}$ (***Source***, T_a)	$\boldsymbol{U^*}$ (***Source***, T_b)	**Final $\boldsymbol{U^*}$ Column**
A	**2/15**	**1/15**	Max(2/15, 1/15) = **2/15**
B	1/15	1/15	1/15
C	1/15	1/15	1/15
D	1/15	1/15	1/15
E	1/15	1/15	1/15
F	1/15	1/15	1/15
G	1/15	1/15	1/15
H	1/15	1/15	1/15
I	2/15	2/15	2/15
J	2/15	2/15	2/15
K	3/15	3/15	3/15
Q	0	2/15	2/15
Y	0	2/15	2/15
Z	0	3/15	3/15
Final Sum	1/15	22/15	**23/15**

Table 17. Hash frequencies of the split files of *Source*.

	$\boldsymbol{S_a}$		$\boldsymbol{S_b}$
Hash	**Frequency**	**Hash**	**Frequency**
A	1	F	1
B	1	G	1
C	1	H	1
D	1	I	2
E	1	J	2
		K	3

nodes is $\frac{7/15}{23/15} = 0.3043$, which matches the JINF(*Source*, *Target*) value in Table 5.

5. Results and Validation

The computing performance of JISPLIT using the ARGO distributed environment was evaluated. The M57 Patents Scenario dataset was used for initial performance testing. The daily images for Terry were 40 GB each and the system images for Pat, Jo and Charlie were 10 GB each. For the 10 GB source image files and 10 GB target image files, the processing time improved around 15%, from an average of 13 minutes

Table 18. JINF(S_a, T_a).

Normalized Hash Frequency	I^*	U^*
A: (1/15, 2/15)	1/15	**2/15**
B: (1/15, 1/15)	1/15	**1/15**
C: (1/15, 1/15)	1/15	**1/15**
D: (1/15, 1/15)	1/15	**1/15**
E: (1/15, 1/15)	1/15	**1/15**
F: (0, 1/15)	0	**1/15**
Sum	**5/15**	

Table 19. JINF(S_b, T_a).

Normalized Hash Frequency	I^*	U^*
A: (0, 2/15)	0	**2/15**
B: (0, 1/15)	0	**1/15**
C: (0, 1/15)	0	**1/15**
D: (0, 1/15)	0	**1/15**
E: (0, 1/15)	0	**1/15**
F: (1/15, 1/15)	1/15	**1/15**
G: (1/15, 0)	0	**1/15**
H: (1/15, 0)	0	**1/15**
I: (2/15, 0)	0	**2/15**
J: (2/15, 0)	0	**2/15**
K: (3/15, 0)	0	**3/15**
Sum	**1/15**	

Table 20. JINF(S_a, T_b).

Normalized Hash Frequency	I^*	U^*
A: (1/15, 0)	0	**1/15**
B: (1/15, 0)	0	**1/15**
C: (1/15, 0)	0	**1/15**
D: (1/15, 0)	0	**1/15**
E: (1/15, 0)	0	**1/15**
G: (0, 1/15)	0	**1/15**
Q: (0, 2/15)	0	**2/15**
Y: (0, 2/15)	0	**2/15**
Z: (0, 3/15)	0	**3/15**
Sum	**0**	

Table 21. JINF(S_b, T_b).

Normalized Hash Frequency	I^*	U^*
F: (1/15, 0)	0	**1/15**
G: (1/15, 1/15)	1/15	**1/15**
H: (1/15, 0)	0	**1/15**
I: (2/15, 0)	0	**2/15**
J: (2/15, 0)	0	**2/15**
K: (3/15, 0)	0	**3/15**
Q: (0, 2/15)	0	**2/15**
Y: (0, 2/15)	0	**2/15**
Z: (0, 3/15)	0	**3/15**
Sum	**1/15**	

to an average of 11 minutes. When the number of split image files was increased to more than a certain value that depended on the original size of the pre-split image file, more time was required for the JINF computations compared with the time required without splitting. The penalty is imposed by the pre- and post-processing steps, which involve multiple file access operations.

Table 22. Final *Intersection** from four ARGO nodes.

Pair of Split Files	$\boldsymbol{I}^*$
$(S_a,\ T_a)$	5/15
$(S_b,\ T_a)$	1/15
$(S_a,\ T_b)$	0/15
$(S_b,\ T_b)$	1/15
Final Sum	**7/15**

Table 23. Final *Union** column from four ARGO nodes.

Hash	$\boldsymbol{U}^*$ $(\boldsymbol{S}_a,\ \boldsymbol{T}_a)$	$\boldsymbol{U}^*$ $(\boldsymbol{S}_b,\ \boldsymbol{T}_a)$	$\boldsymbol{U}^*$ $(\boldsymbol{S}_a,\ \boldsymbol{T}_b)$	$\boldsymbol{U}^*$ $(\boldsymbol{S}_b,\ \boldsymbol{T}_b)$	**Final $\boldsymbol{U}^*$ Column**
A	2/15	2/15	1/15	N/A	2/15
B	1/15	1/15	1/15	N/A	1/15
C	1/15	1/15	1/15	N/A	1/15
D	1/15	1/15	1/15	N/A	1/15
E	1/15	1/15	1/15	N/A	1/15
F	1/15	1/15	1/15	1/15	1/15
G	N/A	1/15	1/15	1/15	1/15
H	N/A	1/15	1/15	1/15	1/15
I	N/A	2/15	2/15	2/15	2/15
J	N/A	2/15	2/15	2/15	2/15
K	N/A	3/15	3/15	3/15	3/15
Q	N/A	N/A	2/15	2/15	2/15
Y	N/A	N/A	2/15	2/15	2/15
Z	N/A	N/A	3/15	3/15	3/15
Final Sum					**23/15**

Table 24. JISPLIT performance using the ARGO platform.

Target Split Files	**Processing Time**
1	22 min 15 sec
3	12 min 41 sec
6	**10 min 49 sec**
9	11 min 57 sec
12	12 min 29 sec
Source: 10 GB, *Target*: 40 GB	

For the 10 GB source image files and 40 GB target image files, the JIS-PLIT computation time was reduced by about half (51%) as shown in Table 24. Computing the JINF values for two huge images is computationally intensive, especially with respect to system memory. However, the results suggest that partitioning the source and target media into smaller files and distributing the computations render the overall times for computing similarity measures feasible. Of course, additional experimentation and testing are necessary to further validate the method.

6. Conclusions

Early triage and prioritization of digital evidence are important tasks performed by digital forensic practitioners. The main goal of this research has been to compute digital media image similarity measures that support efficient triage and association discovery. The proposed method does not replace existing approximate hashing and other techniques; instead, it leverages and potentially augments them. Most existing similarity measures work at the file or object levels. In contrast, the proposed method works at the sector level and is, therefore, robust in the face of deleted and partially-overwritten data.

By splitting digital media images into several files via centralized pre-processing, distributing the separate file computations to nodes and computing a final similarity measure via centralized post-processing, the overall computation time can be reduced up to 51% while achieving the accuracy of the computations performed by a single processor. When the target and source images are much larger, using optimal numbers of splits for the images can yield computational time reductions exceeding 51%. The determination of the optimal number of splits for a digital media image of a given size is a topic for future research; in fact, the optimization may also depend on media content. In any case, high-performance parallel computing infrastructures may be used to implement the proposed method to maximize computational time reductions.

However, there are some caveats. The proposed method may be vulnerable to an adversary who selectively deletes or overwrites content in common with another digital device, plants false fragments to mislead the algorithm and digital forensic practitioners, wipes digital media at a low level and/or encrypts media with unique keys. While these actions would severely limit the effectiveness of the proposed method, the first two require considerable investments of time and skill on the part of an adversary and may also be detectable after the fact. The last two actions are effective, but they would also be obvious to a digital forensic practitioner and would, therefore, be more likely to lead to no results

instead of false results. It should be noted that the proposed method is applicable to compressed and encrypted files as long as the compression methods and encryption keys are the same across systems. Additionally, the method is applicable to damaged media and partially-recovered content because it does not require filesystem information.

References

[1] R. Beverly, S. Garfinkel and G. Cardwell, Forensic carving of network packets and associated data structures, *Digital Investigation*, vol. 8(S), pp. S78–S89, 2011.

[2] P. Bjelland, K. Franke and A. Arnes, Practical use of approximate hash-based matching in digital investigations, *Digital Investigation*, vol. 11(S1), pp. S18–S26, 2014.

[3] B. Bloom, Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM*, vol. 13, pp. 422–426, 1970.

[4] F. Breitinger and H. Baier, Performance issues about context-triggered piecewise hashing, *Proceedings of the International Conference on Digital Forensics and Cyber Crime*, pp. 141–155, 2012.

[5] F. Breitinger and H. Baier, Similarity-preserving hashing: Eligible properties and a new algorithm MRSH-v2, *Proceedings of the Fourth International Conference on Digital Forensics and Cyber Crime*, pp. 167–182, 2012.

[6] F. Breitinger, B. Guttman, M. McCarrin, V. Roussev and D. White, Approximate Matching: Definition and Terminology, NIST Special Publication 800-168, National Institute of Standards and Technologies, Gaithersburg, Maryland, 2014.

[7] S. Bunting and W. Wei, *EnCase Computer Forensics: The Official EnCE: EnCase Certified Examiner Study Guide*, Wiley Publishing, Indianapolis, Indiana, 2006.

[8] S. Garfinkel, Forensic feature extraction and cross-drive analysis, *Digital Investigation*, vol. 3(S), pp. S71–S81, 2006.

[9] S. Garfinkel, Digital media triage with bulk data analysis and `bulk_extractor`, *Computers and Security*, vol. 32, pp. 56–72, 2013.

[10] S. Garfinkel, P. Farrell, V. Roussev and G. Dinolt, Bringing science to digital forensics with standardized forensic corpora, *Digital Investigation*, vol. 6(S), pp. S2–S11, 2009.

[11] S. Garfinkel and M. McCarrin, Hash-based carving: Searching media for complete files and file fragments with sector hashing and `hashdb`, *Digital Investigation*, vol. 14(S1), pp. S95–S105, 2015.

[12] S. Garfinkel, A. Nelson, D. White and V. Roussev, Using purpose-built functions and block hashes to enable small block and sub-file forensics, *Digital Investigation*, vol. 7(S), pp. S13–S23, 2010.

[13] J. Kornblum, Identifying almost identical files using context-triggered piecewise hashing, *Digital Investigation*, vol. 3(S), pp. 91–97, 2006.

[14] M. Lim and J. Jones, A digital media similarity measure for triage of digital forensic evidence, in *Advances in Digital Forensics XVI*, G. Peterson and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 111–135, 2020.

[15] V. Moia, F. Breitinger and M. Henriques, The impact of excluding common blocks in approximate matching, *Computers and Security*, vol. 89, article no. 101676, 2020.

[16] V. Moia and M. Henriques, A comparative analysis of similarity search strategies for digital forensic investigations, *Proceedings of the Thirty-Fifth Brazilian Symposium on Telecommunications and Signal Processing*, pp. 462–466, 2017.

[17] National Institute of Standards and Technology, National Software Reference Library (NSRL), Gaithersburg, Maryland (`www.nsrl.nist.gov`), 2019.

[18] J. Oliver, C. Cheng and Y. Chen, TLSH – A locality sensitive hash, *Proceedings of the Fourth Cybercrime and Trustworthy Computing Workshop*, pp. 7–13, 2013.

[19] H. Parsonage, Computer Forensics Case Assessment and Triage – Some Ideas for Discussion (`computerforensics.parsonage.co.uk/triage/ComputerForensicsCaseAssessmentANDTriageDiscussionPaper.pdf`), 2009.

[20] P. Penrose, W. Buchanan and R. Macfarlane, Fast contraband detection in large capacity disk drives, *Digital Investigation*, vol. 12(S1), pp. S22–S29, 2015.

[21] RCFL National Program Office, Regional Computer Forensics Laboratory Annual Report for Fiscal Year 2017, Quantico, Virginia (`www.rcfl.gov/file-repository/09-rcfl-annual-2017-190130-print-1.pdf/view`), 2017.

[22] R. Real and J. Vargas, The probability basis of Jaccard's index of similarity, *Systematic Biology*, vol. 45(30), pp. 380–385, 1996.

[23] V. Roussev, Building a better similarity trap with statistically improbable features, *Proceedings of the Forty-Second Hawaii International Conference on System Sciences*, 2009.

[24] V. Roussev, Data fingerprinting with similarity digests, in *Advances in Digital Forensics VI*, K. Chow and S. Shenoi (Eds.), Springer, Berlin Heidelberg, Germany, pp. 207–226, 2010.

[25] V. Roussev, Y. Chen, T. Bourg and G. Richard, `md5bloom`: Forensic filesystem hashing revisited, *Digital Investigation*, vol. 3(S), pp. S82–S90, 2006.

[26] V. Roussev, G. Richard and L. Marziale, Multi-resolution similarity hashing, *Digital Investigation*, vol. 4(S), pp. S105–S113, 2007

[27] J. Taguchi, Optimal Sector Sampling for Drive Triage, M.S. Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, California, 2013.

[28] A. Tridgell, `spamsum` (`samba.org/ftp/unpacked/junkcode/spamsum/README`), 2002.

[29] R. Walls, E. Learned-Miller and B. Levine, Forensic triage for mobile phones with DEC0DE, *Proceedings of the Twentieth USENIX Conference on Security*, 2011.

[30] K. Woods, C. Lee, S. Garfinkel, D. Dittrich, A. Russell and K. Kearton, Creating realistic corpora for security and forensic education, *Proceedings of the ADFSL Conference on Digital Forensics, Security and Law*, pp. 123–134, 2011.

[31] J. Young, K. Foster, S. Garfinkel and K. Fairbanks, Distinct sector hashes for target file detection, *IEEE Computer*, vol. 45(12), pp. 28–35, 2012.