The impact of excluding common blocks for approximate matching

Vitor Hugo Galhardo Moia[a], Frank Breitinger[b,*], Marco Aurélio Amaral Henriques[a]

[a] *Department of Computer Engineering and Industrial Automation (DCA), School of Electrical and Computer Engineering (FEEC), University of Campinas, Av Albert Einstein 400, Campinas, SP 13083-852 Brazil*
[b] *Cyber Forensics Research and Education Group (UNHcFREG), Tagliatela College of Engineering, ECECS, University of New Haven, 300 Boston Post Rd., West Haven, CT 06516, United States*

1. Introduction

In digital forensics, approximate matching (AM, Breitinger et al., 2014a) complements traditional hashing for known-file filtering (i.e., whitelisting or blacklisting). These algorithms gained popularity over the past decade with `ssdeep` by Kornblum (2006) being one of the first implementations. Compared to cryptographic hash functions that treat the input as a whole, AM extracts features (i.e., byte sequences), compresses them and returns a final similarity digest (equivalent to a hash value). Consequently, we consider two objects similar if they share identical/common features. While this procedure is well-established, it comes with one drawback – not all features have the same value (from a practitioner's perspective). This was discussed by Foster (2012) and Garfinkel and McCarrin (2015) who showed that many identical blocks happened to be present in many different objects. They referred to these blocks as 'common blocks' or 'non-probative blocks'. For the remainder of this paper we use *common feature* and *common block* as synonyms.

Common blocks To differentiate between the importance of similarity from an investigator perspective, it is important to find a strategy to separate the content generated by users from content generated by applications. Consequently, we define the following terms for this article (based on a discussion by Foster, 2012):

- *User-generated content:* Refers to data created by users (e.g., the text of a Word document) and can be seen as the most relevant similarity between two objects.
- *Template content:* A weaker form of data content is template similarity which is inserted by a user but repeats over many (different) files. An example is a company's Word template containing a header and footer. While this is still relevant information from a practitioner's perspective, it may lead to irrelevant matches.
- *Application-generated content:* Data created by the application. An example is a file-header information needed to access the file which is shared among (almost) all files of the same file type. We claim that this similarity is the least relevant.

Problem definition According to Gutierrez-Villarreal (2015), many application-generated common blocks contain patterns such as repeating *n*-grams or low entropy data (e.g., `NULL` block). However, there are others that are difficult to find, since they do not have any obvious pattern. All these blocks are created by applications as a result of their inner structure and are less relevant from an investigator perspective. To counteract, practitioners are encouraged to ignore low similarity matches, e.g., in case of `sdhash`, Roussev (2011) suggested to only consider matches that have a similarity score of 21 or more. On the other hand, Garfinkel et al. (2010) and

* Corresponding author. new affiliation: Hilti Chair for Data and Application Security at the University of Liechtenstein.

E-mail addresses: vhgmoia@dca.fee.unicamp.br (V.H.G. Moia), Frank.Breitinger@uni.li (F. Breitinger), marco@dca.fee.unicamp.br (M.A.A. Henriques).

URL: https://www.FBreitinger.de (F. Breitinger), http://www.dca.fee.unicamp.br/~marco (M.A.A. Henriques)

Foster (2012) recommended focusing on distinctive blocks. However, there have been no studies discussing how AM tools can be used to remove common blocks (without interpreting the data) and how their removal will impact the similarity score / matches.

Contribution This article analyzes common blocks (i.e., what is a common block) and shows how to use AM to identify and filter out these blocks for a given dataset. Therefore, the first contribution is an open-source application named `NCF_sdhash` (based on `sdhash`). Secondly, we discuss how common blocks are spread across various files (same and different file types), their frequency, and show that removing them can improve the quality of matches. Lastly, we present a threshold to assess if a block should be considered common and measured the effects.

*Overview*The structure of this paper is as follows: The next section provides the background and related work. In Section 3 we outline the research questions, our workflow and details about the implementation (modification of existing algorithms). Experimental results on common features are presented in Section 4, followed by a discussion on the impact of the similarity detection in Section 5. Our results are discussed in Section 6. The last section concludes the paper and outlines next steps.

2. Background and related work

This section focuses on the closely related work of bytewise approximate matching (AM), but will not provide a detailed description of all its concepts. Readers who need additional information are encouraged to read the NIST SP 800-168 which states that it as a "promising technology designed to identify similarities between two digital artifacts" (Breitinger et al., 2014a).

2.1. Bytewise approximate matching algorithms

Over the past years, numerous AM algorithms have been published with different strengths and weaknesses. Some of the more prominent algorithms include: `ssdeep` by Kornblum (2006), `sdhash` by Roussev (2010), `mrsh-v2` by Breitinger and Baier (2013), TLSH by Oliver et al. (2013) or LZJD by Raff and Nicholas (2018).

For this study, we decided to focus on `sdhash` and `mrsh-v2` due to familiarity, their popularity and their differences in feature size (fixed-size vs. variable-size). Additionally, literature shows that both have good detection capabilities (Breitinger and Roussev, 2014; Breitinger et al., 2014b; Roussev, 2011).

sdhash Roussev (2010) proposed `sdhash` which identifies statistically improbable features in objects (i.e., unique features) and uses them to create similarity digests. Specifically, `sdhash` tries to find features (64-byte sequences) "that have the lowest empirical probability of being encountered by chance." Next, each feature is compressed using SHA-1 and stored into a bloom filter (Bloom, 1970). In case a filter reaches its capacity of 160 feature hashes, a new one is created. The final digest is a concatenation of all bloom filters. In order to measure the similarity of two digests, bloom filters are compared against each other using the Hamming distance, and a normalized similarity score (scale: 0 to 100) is returned.

mrsh-v2 Based on the `ssdeep`'s feature extraction process, `mrsh-v2`, proposed by Breitinger and Baier (2013), divides an input into variable-size blocks (*features*) and stores all *features* in a list of bloom filters (adopted from `sdhash`). In detail, the feature extraction process uses a sliding window (a.k.a. rolling hash) of 7 bytes and moves through the object byte-by-byte. Whenever the rolling hash matches a particular value, the end of a feature is found. Next, all extracted features are hashed using an FNV hash function (Noll, 2012), and inserted in a bloom filter. Creating and comparing the digest are identical to `sdhash`.

2.2. Common blocks

The concept of common blocks was first discussed by Garfinkel et al. (2010) where the authors presented hash-based carving for content identification: The idea is to hash hard drive sectors (fixed-size pieces of data between 512 bytes and 4,096 bytes) and try to match a block to a given file. The authors utilized the term *distinct* for the first time referring to blocks that occur only once in their test-corpus. Foster (2012) and Garfinkel and McCarrin (2015) continued the study of hash-based carving and during their tests, they found many common blocks across files which made it difficult to prove the existence of a given file on a media under investigation. The solution adopted by the authors was using *distinct* blocks only.

Besides using a database to filter blocks that appeared several times, Foster (2012) also proposed rules to identify the common / less relevant blocks. The first rule was to ignore blocks with low entropy (i.e., repetition of the same character, `NULL` blocks, etc.). The second rule addressed blocks with repeating *n*-grams. In a follow up work, Garfinkel and McCarrin (2015) recommended additional rules as the entropy calculus was insufficient:

- *Ramp test:* Deals with blocks belonging of Microsoft Office Sector Allocation Tables (SAT);
- *White space test:* Searches and removes blank lines of 100 spaces, each terminated by a newline character (mostly found in JPEG files);
- *The 4-byte histogram test:* 4-byte values, either repeating or alternating 4-byte values, are searched and eliminated. This pattern was found in Apple QuickTime and Microsoft Office file formats.

According to Gutierrez-Villarreal (2015), the rules were redundant, and they proposed replacing them by a single one. Also, by focusing their research on JPEG files only, they found out that using blocks (4,096-byte segment of a file) with an entropy of 10.9 or higher removes many of the common blocks.

Garcia (2018) also explored how common blocks affect the similarity assessment by showing mismatches between fragments of data due to the common structure found among objects. They compared two approaches to extract the common blocks, one using the usual block-based hashing (disk sector level) and another using a rolling hash algorithm (similar to `ssdeep`/`mrsh-v2`) to explore the fragment hash uniqueness on JPEG images and compressed file archives. They report a successful detection of JPEG files inside compressed archives, arguing that many compression algorithms do not compress high entropy data (a previous test on the data segment being compressed is done beforehand). Since JPEG files encompass this category, most content of such files will be stored without significant compression. Later, one can correlate a JPEG file with a compressed archive and yet find some similarity (in case the same image is inside the archive).

However, the aforementioned references have some constraints when it comes to similarity detection. First, some authors restricted their applications to finding exact duplicates while we are interested in finding similar data and small / minimum changes. Second, previous research often focused on relatively large blocks for analysis; however, the larger the block, the more likely to encompasses changes in the object content, resulting in a different hash which prevents finding similar blocks. Lastly, previous work ignores the block alignment problem, i.e., adding / removing a single byte at the beginning of an object will shift the subsequent bytes and change the representing hash. For this reason, we explore the use of Approximate Matching tools (which deal with the aforementioned issues efficiently) to perform object identification (of similar content).

3. Research direction, design decisions and implementation

The impact of common features on the behavior of approximate matching algorithms is not well explored. Therefore, this paper addresses the following research questions:

*RQ*1 What are the common features? How frequently do they appear? How do they spread across various file types?
*RQ*2 How does ignoring common features impact the similarity detection (i.e., number of matches)?
*RQ*3 Is there a clear threshold *N* for which common features are ignored in the dataset at hand?
*RQ*4 How does removing common features affect the runtime efficiency of the algorithm?

It is important to note that we are not interested in measuring the accuracy or detection capabilities of the algorithms but strictly focus on the impact of common features on the similarity matches.

3.1. Procedure overview / workflow

To analyze the impact of common blocks, we performed several experiments. While for RQ1 we compared the behavior of `sdhash` and `mrsh-v2`, the remaining tests concentrated on `sdhash` for three reasons: (a) It appears to be the most widespread approach; (b) it uses a constant and shorter feature size than its competitor; and (c) it produces more features as shown in Section 4.1. Additionally, `sdhash` utilizes the Shannon entropy to exclude undesirable features, eliminating some of the most common blocks by default, e.g., the ones composed by only values of 0s or 1s. Regardless of this choice, we expect similar outcomes for `mrsh-v2` although it has to be validated in future work.

Definition For a better and common understanding, we define the following:

S denotes a dataset of files s and $|S|$ denotes the cardinality.
f denotes a feature where each f belongs to one or more files.
s is a k-tuple of features, e.g., $s = (f_0, f_1, \dots, f_{k-1})$. Note, features are not unique and may repeat in s.
$\mathcal{T}$ denotes a tuple containing all features for all $s \in S$ (order not important). Note, features are not unique and may repeat.
$\mathcal{F}$ denotes the set containing all features from $\mathcal{T}$. Recall: sets only contain unique elements.
$tf(f, \mathcal{T})$ denotes the feature frequency[1] and is the raw count of f in $\mathcal{T}$.
$itf(f, S)$ denotes the inverse term frequency[1] and is the raw count of $s \in S$ that contain f, i.e., the number of files containing f one or more times.
Common feature f is a feature where $itf(f, S) > N$ where N denotes a threshold.

Procedure for RQ1 In order to understand the spreading of common features, we modified `sdhash` and `mrsh-v2` to store the extracted features in a database. Next, we executed both modified implementations on S and counted the frequencies of each feature. Lastly, we analyzed the frequencies which allowed us to compare `sdhash` and `mrsh-v2` as well as provided an overview of how many common features / unique features exist in S. Results are discussed in Section 4.1. Additionally, we manually analyzed some common features which are highlighted in Section 4.2.

Procedure for RQ2 To measure the impact of ignoring common features on similarity matches (number of matches), we compared the matching behavior of `sdhash` and `NCF_sdhash` for various N.

[1] Terms are borrowed from information retrieval field as they are similar.

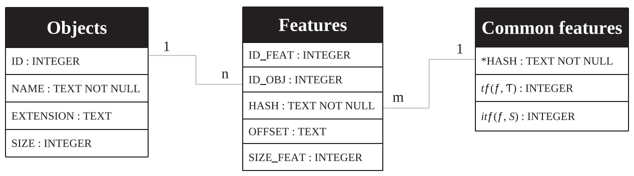


Fig. 1. SQLite database tables used to store all features (distinct and common) and related metadata.

If a feature is common according to our definition, it will be ignored during processing.

All ignored features are not represented in the similarity digest. Findings are summarized in Section 5.1.

Procedure for RQ3 To identify a valid threshold, we used some pre-defined N-values and manually inspected the matches. This allowed us to assess the best N for S as well as various file types. Results are presented in Section 5.1.

Procedure for RQ4 Our modified version is compared to the original version with respect to runtime efficiency, i.e., the time to generate and compare digests. Results are outlined in Section 5.6.

Differentiation of similarity To assess whether a match was related to user-generated content, application-generated content, or template similarity, we manually investigated matches. When files had no visual similarity, i.e., no common text, picture, table, or other user-generated elements, we classified it as application-generated content similarity. A match was considered template similarity when the same layout repeated over several files, but their content was different. For instance, two `html` pages, 'contact us' and 'our organization', where both files had identical colors, elements disposition, menu bar, headlines, logo, etc., but their content was different. In this work, we focus on user-generated content which will be considered true positives; template and application-generated content are considered false positives.

3.2. Database implementation

We chose SQLite as the relational database management system for its simplicity and being open source[2] In our experiment, three tables were created to store all the extracted feature hashes including related metadata such as offsets. An overview is depicted in Fig. 1. The field highlight with an asterisk (*) is indexed (SQLite, 2019a).

The *Objects*-table contains information about the processed file such as its name (path), size (bytes), and extension (file type). The *Features*-table stores data about the features themselves, e.g., feature hash, its offset (position where the feature content is stored in the file), and the size of the feature. Note, a file consists of many features and each one has a distinct entry in the features table (even if the same feature occurs multiple times, they all have different offsets).

The final table, *Common features*, acts as counter storage for each feature f and contains $tf(f_i, \mathcal{T})$ for $0 < i \leq |\mathcal{T}|$ as well as $itf(f, S)$ for all $f \in \mathcal{F}$. For instance, for a set $S = \{A, B\}$, if feature f_1 occurs 10 times in file A and twice in file B, then $tf(f_1, \mathcal{T}) = 12$ and $itf(f_1, S) = 2$.

3.3. Implementation changes to existing tools

To perform our assessment, both `sdhash` and `mrsh-v2` required some modification to cope with a database. The database

[2] SQLite is slower than other databases which impairs runtime efficiency tests. If the focus is efficiency, a custom build storage solution as pointed out by Foster (2012) should be preferred.

Table 1
t5-corpus file type statistics.

	html	text	pdf	doc	ppt	xls	jpg	gif
# of files	1093	711	1073	533	368	250	362	67
Avg. size (kb)	66	345	590	433	1003	1164	156	218

Table 2
Feature statistics across S.

Parameter	`sdhash`	`mrsh-v2`
$\lvert\mathcal{T}\rvert$	31,387,592	8,842,032
$\lvert\mathcal{F}\rvert$	27,203,732	8,049,461
$max(tf(f,\mathcal{T}))$	153,037	8141
$max(itf(f,S))$	843	790

Table 3
Feature frequencies across S.

Condition	`sdhash`	`mrsh-v2`
$count(f \mid tf(f,\mathcal{T}) = X)$		
$X = 1$	25,861,720	7,751,709
$X = 2$	942,182	225,740
$X = 3$	145,236	28,392
$count(f \mid tf(f,\mathcal{T}) > X)$		
$X > 3$	254,594	43,620
$X > 5$	124,413	22,100
$X > 10$	39,069	8044
$X > 20$	16,840	3318
$X > 50$	5986	1232
$X > 100$	2663	526
$X > 200$	1124	231
$X > 400$	471	81
$X > 800$	185	32
$X > 2,000$	63	11
$X > 10,000$	6	0
$count(f \mid f \in \mathcal{F} \wedge itf(f,S) = N)$		
$N = 1$	26,467,390	7,886,026
$N = 2$	567,267	140,064
$N = 3$	68,719	12,251
$count(f \mid f \in \mathcal{F} \wedge itf(f,S) > N)$		
$N > 3$	100,356	11,120
$N > 5$	43,845	5419
$N > 10$	5386	1356
$N > 20$	1676	488
$N > 50$	579	200
$N > 100$	287	115
$N > 200$	115	55
$N > 400$	17	10

and applications can be downloaded from github (programing language C++): https://github.com/regras/cbamf. Specifically, the following changes were made:

DB creation: The feature extraction process of each tool was modified to insert the features and required metadata into the database. Modifications were tested manually to verify the correctness of the new version.

No common feature (NCF): The second modification was slightly more complex as it performs queries in the database (db). In other words, before proceeding with an identified feature, the db is queried to check if it is a common feature, where 'common' depends on the user setting N. If a feature is common, it is discarded; otherwise, it will be further processed and added to the similarity digest. The new versions are named `NCF_sdhash` (based on `sdhash` 3.4) and `NCF_mrsh-v2`.

Lastly, we also decided to use FNV-1a (64 bits) for the feature hashing algorithm as suggested by Kameyama et al. (2018) to improve the runtime efficiency without affecting the tool's precision.

4. Experimental results on common features

The dataset S used for the experiments is the *t5-corpus* (Roussev, 2011) which established itself as a default set in this domain. The corpus is a collection of real-world files containing various file types having a total of 4457 objects (1.78 GiB) as summarized in Table 1.

For validation, we performed a manual comparison of several matches to classify them according to their similarity type (user-generated content, application-generated content, or template content). Since the number of matches is too large, we only analyzed a sample of the them using either the appropriate software (e.g., MS office, browser etc.) or `Bless`[3] editor for binary. These results can be found on our github page (see Section 3.3).

4.1. Common blocks overview in t5-corpus

Table 2 shows the statistics for the extracted features in S separated by tool. Note, $max(tf(f,\mathcal{T}))$ returns the number of the most frequently counted feature while $max(itf(f,S))$ for all $f \in \mathcal{F}$ looks for the most common (wide spread) feature and returns the number of files sharing it (e.g., the last row in Table 2 indicates that there is one feature that was found in 843 different files).

As shown, `sdhash` extracted more features than `mrsh-v2` (about 3.5 times), which was expected since `mrsh-v2` comes with a higher compression rate: `sdhash` uses fixed-size features of 64 bytes while `mrsh-v2` has variable-size features. In our experiment, we verified that the average feature size for `mrsh-v2` was 215.3 bytes.

The frequencies of the extracted features are presented in Table 3, where the left column shows the analyzed condition followed by the two algorithms on the right. For instance, row $X > 10$ means that `sdhash` found 39,069 features that occurred more than ten times in $\mathcal{T}$. In contrast, the second part of the table focuses on the number of files containing particular features, e.g., last row indicates that for `mrsh-v2` we found ten features that were in more than 400 files.

The results also showed that a significant number of features repeats frequently, e.g., `sdhash` found 2663 features that repeated more than 100 times (this also means many features repeat within the same file). As indicated by the last part of the table, some features were widely spread among files, e.g., 579 features appeared in more than 50 files.

4.2. Most common features for each file type

Table 4 shows the features that repeated the most across different files of the same type and includes the feature's FNV-1a hash, the number of files, and a brief description of the feature's content. The `doc` feature was related to necessary structural information common in Microsoft Office Word documents and is illustrated in Fig. 2 (highlighted area). It corresponds to the final part of a stream name followed by some setting and padding information. The `doc` feature was found in 442 files which is 83% of all `doc`'s (533) in *t5-corpus*. Although not all `doc`'s had this particular feature, variations of it were found in other files (see Fig. 3). While a different feature was selected, it belongs to the same file structure information. We believe the same happened in the remaining 91 *doc* files, where some specific changes affected the feature selection process. The most common feature of `pdf`, `jpg` and `gif` files were related to color space information. In the case of `html` files, the common feature was associated with a well-known piece of java script code.

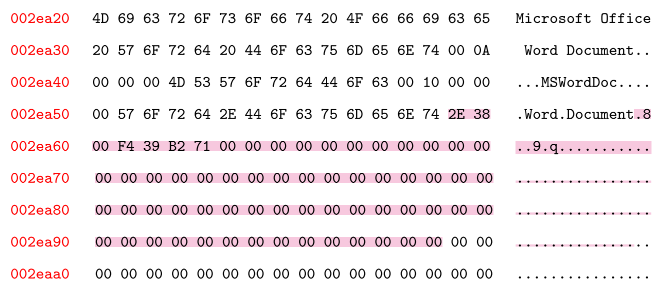
[3] https://github.com/bwrsandman/Bless (last accessed 2019-10-21).

Table 4
Most repeated features per file type and their content.

File type	FNV-1a hash	*itf*(*f*, *S*) for all *s* of the same type	Feature content
`doc`	`c5e7aeb2482c56c0`	442 / 533	Necessary stream of compound files, specific of Microsoft Office Word documents.
`ppt`	`ef9a5a76d0df0c16`	357 / 368	Part of a document summary information stream with application defined properties.
`pdf`	`d5fb4ee41392d833`	347 / 1073	Piece of an indirect object of a pdf stream, belonging to RGB color space.
`xls`	`b3310ce89e000aa4`	226 / 250	Font specification.
`jpeg`	`f0a05cdcac5796d4`	108 / 362	RGB color palette.
`html`	`cbac5aaf609ccf54`	61 / 1093	Sample of a well-known piece of java script code to make web pages have rollover images.
`text`	`69c06bea6c3a3f10`	18 / 711	Part of a template content.
`gif`	`c91811dfd69ce32b`	5 / 67	Related to a global color table, which is a sequence of bytes representing RGB color triplets.

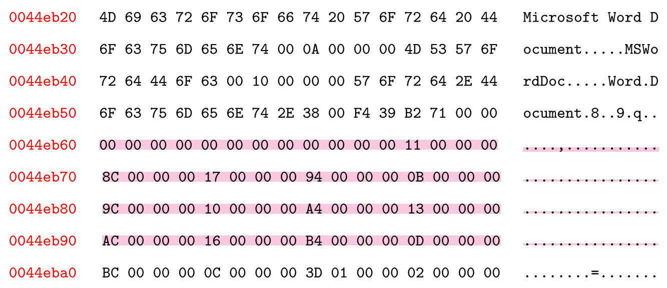
Table 5
Samples of common features that appeared on different file types.

f	$tf(f, \mathcal{T})$	*itf*(*f*, *S*)	*itf*(*f*, *S*) separated by file types
`5d60dae303171ac8`	1014	843	`doc` (404), `ppt` (265), `xls` (174)
`eee894cd42564cc9`	634	582	`doc` (286), `ppt` (295), `xls` (1)
`c02fde95428198dc`	540	531	`doc` (186), `ppt` (267), `xls` (78)
`ef9a5a76d0df0c16`	691	484	`doc` (92), `ppt` (357), `xls` (35)
`c5e7aeb2482c56c0`	468	467	`doc` (443), `ppt` (2), `xls` (22)
`d5fb4ee41392d833`	615	457	`doc` (14), `jpg` (49), `pdf` (347), `ppt` (47)
`536857624aa47c38`	451	437	`doc` (122), `ppt` (243), `xls` (72)
`ce5c0a5b70cca619`	3608	402	`doc` (31), `jpg` (108), `pdf` (185), `ppt` (76), `xls` (2)
`3c0dc7d9b4044951`	224	220	`doc` (6), `xls` (214)
`1a5918d3d2ad6ffe`	228	203	`doc` (3), `ppt` (200)
`87b92f4dc954a121`	193	116	`doc` (14), `jpg` (49), `pdf` (6), `ppt` (47)



```
002ea20   4D 69 63 72 6F 73 6F 66 74 20 4F 66 66 69 63 65   Microsoft Office
002ea30   20 57 6F 72 64 20 44 6F 63 75 6D 65 6E 74 00 0A    Word Document..
002ea40   00 00 00 4D 53 57 6F 72 64 44 6F 63 00 10 00 00   ...MSWordDoc....
002ea50   00 57 6F 72 64 2E 44 6F 63 75 6D 65 6E 74 2E 38   .Word.Document.8
002ea60   00 F4 39 B2 71 00 00 00 00 00 00 00 00 00 00 00   ..9.q...........
002ea70   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
002ea80   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
002ea90   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
002eaa0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Fig. 2. Excerpt of the `000260.doc` file corresponding to the most common feature over this kind of format. The Bless hexadecimal editor is used to show the binary structure of the file. The highlighted area represents the feature.



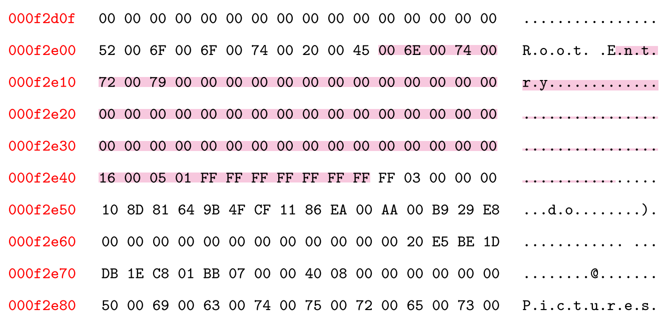
```
0044eb20   4D 69 63 72 6F 73 6F 66 74 20 57 6F 72 64 20 44   Microsoft Word D
0044eb30   6F 63 75 6D 65 6E 74 00 0A 00 00 00 4D 53 57 6F   ocument.....MSWo
0044eb40   72 64 44 6F 63 00 10 00 00 00 57 6F 72 64 2E 44   rdDoc.....Word.D
0044eb50   6F 63 75 6D 65 6E 74 2E 38 00 F4 39 B2 71 00 00   ocument.8..9.q..
0044eb60   00 00 00 00 00 00 00 00 00 00 00 00 11 00 00 00   ....,...........
0044eb70   8C 00 00 00 17 00 00 00 94 00 00 00 0B 00 00 00   ................
0044eb80   9C 00 00 00 10 00 00 00 A4 00 00 00 13 00 00 00   ................
0044eb90   AC 00 00 00 16 00 00 00 B4 00 00 00 0D 00 00 00   ................
0044eba0   BC 00 00 00 0C 00 00 00 3D 01 00 00 02 00 00 00   ........=.......
```

Fig. 3. Excerpt of the `004964.doc` file corresponding to a feature similar to the one presented in Fig. 2. The Bless hexadecimal editor is used to show the binary structure of the file. The highlighted area represents the feature.

For `text`, we found that all 18 files shared the same template, but they differed in content.

4.3. Common features across different file types

This section presents common features that were found across different file types; results are summarized in Table 5. Many features appeared across different files, especially among `doc`, `ppt`, and `xls`, which are all compound files. For instance, $hash(f)$ = `536857624aa47c38` was part of a sector allocation table (SAT) data structure of compound files (Rentz, 2007). However,



```
000f2d0f   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000f2e00   52 00 6F 00 6F 00 74 00 20 00 45 00 6E 00 74 00   R.o.o.t. .E.n.t.
000f2e10   72 00 79 00 00 00 00 00 00 00 00 00 00 00 00 00   r.y.............
000f2e20   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000f2e30   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000f2e40   16 00 05 01 FF FF FF FF FF FF FF FF 03 00 00 00   ................
000f2e50   10 8D 81 64 9B 4F CF 11 86 EA 00 AA 00 B9 29 E8   ...d.o........).
000f2e60   00 00 00 00 00 00 00 00 00 00 00 00 20 E5 BE 1D   ............ ...
000f2e70   DB 1E C8 01 BB 07 00 00 40 08 00 00 00 00 00 00   ........@.......
000f2e80   50 00 69 00 63 00 74 00 75 00 72 00 65 00 73 00   P.i.c.t.u.r.e.s.
```

Fig. 4. Snippet of the `001025.ppt` showing the first occurrence of f = `5d60dae303171ac8` at offset 0xF2E0B.

we also found features shared by compound, `jpg`, and `pdf` files like `d5fb4ee41392d833`, which was part of a color space object (RGB). This was due to embedding objects like images into other file types (in our example all identified objects contained pictures). We also found features related to font specifications shared by different file types, too.

We also analyzed instances where features repeated within the same file. For instance, `5d60dae303171ac8` occurred 1014 times in S (in 843 different files). One of the files containing it was *001025.ppt*. The feature was part of a compound file data structure related to a root directory entry of a stream, where the string *Root Entry* had to be present. The `ppt` contained four similar snippets (see Fig. 4) at different offsets: 0xF2E0B, 0xF360B, 0xF6E0B, and F720B. Besides the feature, the majority of the bytes shown in the figure are identical among all four offsets.

The feature that repeated the most in S ($max(freq(\mathcal{F}))$ = 153,037) is shown in Fig. 5 and belongs to template similarity in `pdf`s. For instance, it repeats 16,092 times in `001958.pdf`; another 144 files shared this feature one or more times. It is part of the cross-reference table (xref), which contains the references to all the objects in a `pdf` document. An object, in this case, is represented by one entry of 20 bytes, consisting of an offset (first 10 bytes), a space separator, the object generation number, another space separator, and a letter ‘f’ or ‘n’ indicating whether the ob-

```
000bdf40  66 0D 0A 30 30 30 30 30 30 30 30 30 30 20 36 35   f..0000000000 65
000bdf50  35 33 35 20 66 0D 0A 30 30 30 30 30 30 30 30 30   535 f..000000000
000bdf60  30 20 36 35 35 33 35 20 66 0D 0A 30 30 30 30 30   0 65535 f..00000
000bdf70  30 30 30 30 30 20 36 35 35 33 35 20 66 0D 0A 30   00000 65535 f..0
000bdf80  30 30 30 30 30 30 30 30 30 20 36 35 35 33 35 20   000000000 65535
000bdf90  66 0D 0A 30 30 30 30 30 30 30 30 30 30 20 36 35   f..0000000000 65
000bdfa0  35 33 35 20 66 0D 0A 30 30 30 30 30 30 30 30 30   535 f..000000000
```

Fig. 5. Feature `aecec3a6185401f1` from `001958.pdf` that was found most frequently (153,037 times) in `pdf`s.

Table 6
Number of file matches by score range using `sdhash` and `NCF_sdhash` for ALL file types, discarding common features with occurrences $> N$.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
$= 1$	2992	65	93	152	195	253	311
≥ 1	9220	409	622	1188	1541	2123	2371
≥ 10	1795	241	356	745	963	1249	1262
≥ 21	1038	181	267	563	799	925*	925*
≥ 50	459	79	114	237	414	475	472
≥ 90	86	20	21	55	58	85	85
$= 100$	18	6	6	15	15	30	30

*Note, same numbers in two columns do not mean that the sets of matches are identical.

ject was free or in use. The final two bytes are the characters CRLF (`0x0D0A`) (Adobe-Systems, 2008).

5. Impact on similarity detection

This section highlights the impact of removing the common features from the similarity digests. Thus, when comparing two digests, common features will not impact the similarity score.

5.1. Summary of number of matches in the dataset

Table 6 shows the number of file matches performing an all-against-all comparison (excluding self-comparisons) on *S* (*t5-corpus*) using `sdhash` and `NCF_sdhash` which equals $(4457 * 4456/2 =)\ 9,930,196$ comparisons.

Column one is the range of the similarity score; column two the number of file matches for `sdhash` followed by the number of file matches for `NCF_sdhash` for various *N*.

Note, the implementations return scores ranging from 0 to 100. However, we omit 0 scores as we are only interested in comparisons with some level of similarity.

There was a significant reduction in the number of matches when excluding common features: `sdhash` returned a total of 9220 matches (score ≥ 1), while dropping common features reduced it to 409 (-95%), 1188 (-87%), and 2371 (-74%) for *N* equal to 3, 10, and 100, respectively. As expected, the more restrictive we were (smaller *N*), the lower the number of matches.

To better understand the results, the upcoming sections focus on each file type. Specifically, we compare all files to a given type against *S*.

5.2. Compound file type (doc, ppt, xls)

Compound files are known for storing numerous files and streams within a single file, in a hierarchical way, similar to a file system. The content streams are further divided into small blocks of data (called sectors) used to store both user and internal control data. The entire file consists of a header and a list of all sectors. Each sector has a fixed-size (usually 512 bytes) defined in the header (Rentz, 2007).

Table 7
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `doc` files, discarding common features with occurrences $> N$.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
$= 1$	1082	7	6	10	9	19	44
≥ 1	4095	49	81	97	100	135	194
≥ 10	607	33	62	62	61	69	72
≥ 21	166	25	48	47	49	50	50
≥ 50	15	6	13	12	15	14	14
≥ 90	0	0	0	0	0	0	0

Table 8
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `xls` files, discarding common features with occurrences $> N$.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
$= 1$	42	9	13	35	26	21	25
≥ 1	133	27	54	98	95	106	108
≥ 10	36	7	16	36	36	37	37
≥ 21	16	4	9	16	16	16	16
≥ 50	2	0	0	1	2	2	2
≥ 90	0	0	0	0	0	0	0

Table 9
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `ppt` files, discarding common features with occurrences $> N$.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
$= 1$	1252	10	23	29	32	44	44
≥ 1	1952	70	90	112	115	169	171
≥ 10	55	36	36	37	36	37	39
≥ 21	23	22	23	24	24	23	23
≥ 50	8	9	8	8	7	7	8
≥ 90	2	2	2	2	2	2	2
$= 100$	1	1	1	1	1	1	1

Tables 7–9 summarize the findings. Our results show a similar behavior among all three types: lots of matches for the original `sdhash` for low score ranges and a significant reduction when removing the common features. For the upcoming detailed analysis, we focused on `doc`'s but expect a similar behavior for the others.

Roussev (2011) mentioned that for compound types, matches with scores below 21 contain many false positives and should be neglected. Thus, we focused on the 166 matches with a score ≥ 21. After performing a manual comparison of all matches, we conclude that 120 cases were not similar regarding user-generated content and 28 matches were classified as template similarity. The remaining 18 matches were similar in terms of user-generated content.

When considering the results of `NCF_sdhash`, we see a significant reduction; for $N = 3$ we only had 25 matches, in which four were classified as application-generated content similarity and five as template similarity. The remaining 16 comparisons were user-generated content similarity (compared to `sdhash` two were missed). Increasing $N = 5$ resulted in 48 matches and also included 16 user-generated content matches plus four application-generated content ones; the other matches were related to template similarity. For $N \in \{10, 20, 50, 100\}$ we had similar results as $N = 5$; all missed two matches.

Focusing on matches with scores < 21 for `sdhash`, we randomly sampled 20 out of the 3929 (4,095-166) total. 18 had no user-generated content similarity, and the remaining two had template similarity. On the other hand, `NCF_sdhash` returned 24 (49-25) matches: Seven template similarity, eight application-generated, and nine user-generated content matches. Out of the

Table 10
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `pdf` files, discarding common features with occurrences $>$ N.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
= 1	492	33	39	62	104	128	163
≥ 1	1684	91	125	286	393	488	674
≥ 10	191	20	33	109	117	161	171
≥ 21	92	12	21	76	76	88	88
≥ 50	45	4	7	37	37	47	45
≥ 90	31	3	3	27	17	29	29
= 100	0	0	0	0	0	0	0

user-generated content case, three matches were cross file type comparisons which were matches between different file types (e.g., `doc` vs. `html`); the other six combined similar content as well as template similarity. The 24 matches for `NCF_sdhash` did not include the two matches identified through random sampling for `sdhash`.

In other words, removing common features reduced the number of matches *significantly*. For instance, for scores ≥ 1, `sdhash` had 4095 cases while `NCF_sdhash` returned between 49 matches ($N = 3$; best case) and 194 ($N = 100$; worst case), a reduction of 99% and 95%, respectively. The reduction dropped to values between 85% and 70% when considering only matches with scores ≥ 21 (the recommended value for `sdhash`). However, 70% is still a significant reduction considering the digital forensic examiner may have to compare matches manually.

For the compound file type, using $N = 3$ revealed the best results. We argue that a significant reduction in the number of matches outweighs the two additional matches (user-generated content) identified by `sdhash`.

Remark Compound files usually also have a minimum size; for the types discussed here it is three sectors / 1536 bytes). Consequently, an empty document will have at least 1536 bytes of structural information which impacts the final similarity score especially for small files: Since small files have more application-generated than user-generated content, many (undesirable) matches may occur.

5.3. PDFs

Out of the 92 matches for `sdhash` ≥ 21 in Table 10, only 38 pairs could be manually evaluated; 11 matches included user-generated content. In the remaining pairs at least one of the files was corrupted[4]. A closer look at the application-generated content matches (27) revealed that the majority of features were related to color information (e.g., `d9e1c063e9c0ba1c`); we also found some features (`ca80692484c3235c`) corresponding to a `pdf` object containing Adobe's Extensible Metadata Platform (XMP) data, a package to add metadata to images (but also other media files). Another feature (`4c815162434ce18d`) contained bytes of the XMP data object and lots of black spaces.

The impact of removing the common features was again significant. `NCF_sdhash` with $N = 3$ returned exactly the 11 user-generated content matches found by `sdhash` plus one extra pair that `sdhash` scored with 20. Raising $N = 5$ resulted in 21 matches; the additional nine matches were related to application-generated similarity. Thus, for `pdf`s, $N = 3$ worked perfectly.

[4] We found that despite the `.pdf` extension, these objects are not *pdf* files but edited `html` files with a few line feed (hex:0A) and space (hex:20) characters inserted into their beginning. The list of these objects can be found: (https://github.com/regras/cbamf).

Table 11
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `text` files, discarding common features with occurrences $>$ N.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
= 1	14	6	8	13	7	9	8
≥ 1	57	35	39	45	41	42	41
≥ 5	27	26	27	26	26	26	26
≥ 10	25	23	25	25	25	25	25
≥ 21	20	18	19	19	19	19	19
≥ 50	5	6	6	6	6	6	6
≥ 90	0	0	0	0	0	0	0

Table 12
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `html` files, discarding common features with occurrences $>$ N.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
= 1	43	16	23	24	33	53	55
≥ 1	1215	185	281	617	857	1275	1278
≥ 5	1052	152	227	568	787	1099	1099
≥ 10	936	139	202	510	721	976	976
≥ 21	759	111	158	409	643	765	765
≥ 50	394	57	83	175	349	409	407
≥ 90	56	15	16	26	39	57	57
= 100	17	5	5	14	14	29	29

5.4. TEXT and HTML

`Text` and `html` are flat file types that do not contain application-generated information. However, `html` files may contain markup elements or scripting languages (e.g., java script). As a consequence, Roussev (2011) suggested using matches with a score ≥ 5 (compared to ≥ 21).

An overview of the `text` results is given in Table 11. The 27 matches found by `sdhash` consisted of 25 user-generated matches, one related to template similarity and the last one was not actually a text file (although file extension was `text`, the file was a `doc` and matched another `doc`). Running `NCF_sdhash` and $N = 3$ returned 26 matches, where 23 fell into user-generated content similarity, two were related to template similarity, and the last one was a cross file type comparison with one of the corrupted `pdf` files. Two matches were missed: in one case there were many changes throughout the file; in the other the `text` file was converted into `html`. Setting $N = 5$ or higher solved this problem; all 25 user-generated matches were found. In summary: the exclusion of the common features for `text` was less effective than other file types but did harm for $N \geq 5$.

With respect to `html`, results were different and are shown in Table 12. `sdhash` returned 1052 matches with scores ≥ 5. In comparison, `NCF_sdhash` reduced this number for small *N*'s but found more matches for higher *N*'s (discussed later). Due to a large number of matches, we randomly sampled 30 cases in each analysis. `sdhash` had no user-generated content match in all 30 samples; 28 were template similarity cases, and two showed application-generated content. `NCF_sdhash` with $N = 3$ had six user-generated content cases and 24 template similarity (for another 30 samples - all different from the first 30). For $N = 5$ and 30 new samples, 25 cases showed template similarity, four cases application-generated content and one user-generated content similarity (an embedded object with minor changes).

The tool `sdhash` found all user-generated content matches present in the 90 samples, while `NCF_sdhash` (for $N = \{3, 5\}$) missed one match.

Table 13
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `jpeg` files, discarding common features with occurrences $>$ N.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
$= 1$	547	3	2	3	2	8	10
≥ 1	907	4	3	5	4	63	69
≥ 10	24	0	0	0	0	0	0
≥ 21	0	0	0	0	0	0	0

Table 14
Number of file matches by score range using `sdhash` and `NCF_sdhash` for `gif` files, discarding common features with occurrences $>$ N.

Score	sdhash	NCF_sdhash for *N*					
		3	5	10	20	50	100
$= 1$	1	0	0	1	1	1	0
≥ 1	1	0	0	1	1	1	0
≥ 10	0	0	0	0	0	0	0

To conclude: $N = 3$ had the best cost/benefit scenario for `html` files.

More `NCF_sdhash` matches than `sdhash` Table 12 suggests that the number of results increased for `NCF_sdhash` for some N's (e.g., compare column `sdhash` vs. $N = \{50, 100\}$). This behavior was the opposite to other file types which is due to hash collision (`sdhash` uses SHA-1 for hashing the features while `NCF_sdhash` uses FNV-1a). We confirmed this by looking into our database: The most frequent feature has 61 different occurrences ($\omega = 61$). Consequently, `NCF_sdhash` for $N = 100$ should have identical results to `sdhash` since no feature was removed.

For instance, the last row ($= 100$) shows 17 matches for `sdhash` but 29 for `NCF_sdhash` with $N = \{50, 100\}$. The 12 new matching pairs involved two files (`001326.html` and `003467.html`) and received `sdhash` scores between 92 and 98. The number of features extracted was 661 by `sdhash` and 660 by `NCF_sdhash` (for $N > 50$). The difference on both was related to a single feature that had a collision with other feature when inserted into the digest (bloom filter) of the file. Since this feature was different from the ones of the other files under comparison, removing it made the similarity score increase to 100. It is important to mention that changing a single file may impact several comparisons, as shown by these two files that affected 12 matches. For `NCF_sdhash` with $N = 50$ we had the same situation, but some other comparisons were affected since in this case two `html` features were removed as they repeat 55 and 61 times each.

5.5. JPEG and GIF

Bytewise approximate matching algorithms work less well on images (but are good for detecting embedded images in compound files). Consequently, no matches were revealed for scores above 21, as shown in Tables 13 and 14. We investigated the 24 `jpg` pairs having a `sdhash` score >= 10 where two matches showed some similarity: a similar `jpg` image was found inside a `ppt` file. However, the pictures were not identical, and thus we categorized it as application-generated content similarity. The other 22 cases had no visual similarity.

Selecting 30 random samples from the 883 (907-24) matches (with score < 10), we found one case (score 3) where one picture was a scaled version of the other. These two `jpgs` had, 555 and 1497 features, respectively. The overlap was 129 features, where 16 were exclusive to these two `jpgs` ($\omega = 2$ for these 16 f) and related to header / EXIF information. Specifically, `97e7cd722414356e` was part of the JFIF header of both files, and features `54fbfe2c46f4bed8` and `581f94d602fd9ac6` were related to EXIF data, such as date time and camera information. This match was hard to classify: on the one hand it is application-generated content; on the other hand it ties it to a particular user. More details are discussed in Section 6: *Differentiating between user-generated and other types of content*. When looking at the scores < 10 and $3 \leq N \leq 20$, only one match had similarity related to template (pictures shared the same background, fonts, and colors). We did not analyze the matches for $N = \{50, 100\}$.

`gifs` had a similar behavior and only one match (`gif` to `text`) was found by `sdhash` which was a false positive. `NCF_sdhash` also had a false positive (with an `html` file) for some N-values.

For both file types, `sdhash` worked suboptimal which is a known challenge for bytewise approximate matching. However, as shown by our results, `NCF_sdhash` reduced the matches without impacting the quality of the results. $N = 3$ worked reliably in both cases.

5.6. Performance test

Besides evaluating how the similarity was affected by removing the common features, we also measured the runtime efficiency as well as the compression rate for `NCF_sdhash` and compared to `sdhash`. The results are shown in Table 15. Here we present tests measuring the time taken for creating digests for each file in S and then performing an all-against-all comparison (excluding the self-comparison) using both tools. We also measured the final digest size of all files of the set to examine how the compression rate was affected. In our tests, we did not include the offline steps (i.e., time to extract all features and insert them into the db). Although this task requires a long time, it does not impact the investigation as it can be done offline / at any time. It is also done only once.

The test environment is an i7-5500U CPU @2.40 GHz processor, 8 GB of memory, 1 TB SATA 3 Gb/s hard disk drive (5,400 rpm), and NVIDIA GeForce 920M, running an Elementary OS 0.4.1 Loki 64-bit (built on Ubuntu 16.04.2 LTS).

Each experiment ran 20 times; results were averaged. The cache was cleared every run to prevent falsification of the results.

We also turned off all unneeded system services and stop unnecessary applications (Kim et al., 2012).

The measurement was done using Linux `time` command (`sys` + `user` times).

As shown by our results, `NCF_sdhash` is about 24% to 26% slower with respect to digest generation time which is related to query the db (verifying if a feature is common or not). As expected, the runtime is independent of N as shown in the Table 15. We also measured the time to perform an all-against-all comparison utilizing the existing db. `NCF_sdhash` ($N = 3$) was slightly faster than `sdhash` which is due to the removed features resulting in fewer bloom filters (note the code related to the comparison function did not change). On the other hand, higher N-values for `NCF_sdhash` removed fewer features and thus have similar times than `sdhash`. The last parameter measured is the digest size generated for S. Here, `NCF_sdhash` was superior since it removed the common features, resulting in smaller digests. The lower the N-value, the larger the reduction achieved.

Remark It is important to highlight that the only optimization performed in the `NCF_sdhash` code was the creation of database indexes for the common feature table. As mentioned before, the creation of the common feature database is required only once and can be done offline; afterwards it can be used for all investigations. `NCF_sdhash`'s bottleneck (when creating digests) are SQLite queries. The complexity for verifying whether a feature is common or not is $O(log(|\mathcal{T}|))$ (when using database indexes, SQLite, 2019b). Other optimizations focus on SQLite itself as discussed by Purohith et al. (2017). On the other hand, one could move to a cus-

Table 15
Runtime and digest size of `sdhash` and `NCF_sdhash` for *t5-corpus*; includes the measurement and standard deviation (±).

	sdhash	NCF_sdhash for *N*		
		3	10	100
Digest generation (sec)	87 s ± 0.92	108 s ± 0.42	110 s ± 0.41	110 s ± 0.41
All-against-all comparison (sec)	433 s ± 11.62	413 s ± 11.52	452 s ± 12.53	458 s ± 15.82
Digest size (bytes)	64,321,035	62,594,719	63,786,954	64,138,018

tomized storage solution, which should improve the query performance (Foster, 2012). With respect to a database update: The features of the new objects are extracted (using the tool) and inserted into the database which is not time critical.

6. Discussion

Based on the experiments described in the previous section, we will discuss here the lessons learned and the impacts of removing the common features, starting with the research questions.

RQ1. What are the common features? How frequently do they appear? How do they spread across file types?

Foster (2012) stated that most files are made up of distinct blocks/features that identify a specific file which also holds for our S: given the 31 million features, only 0.8% were common (for $N = 3$). However, as pointed out by Garfinkel and McCarrin (2015), features that appear to be unique could be uncommon if the dataset is expanded.

The common blocks spread widely among the same or different file types although different file types showed different behavior. For instance, compound file types (`doc`, `ppt`, `xls`) have a high degree of 'default similarity' as they share a similar internal structure which is in contrast to flat types (`text`, `html`) and compressed types (`jpg`, `gif`). For

compound types the proposed `NCF_sdhash` tool can reduce the number of false positives significantly (similar results were obtained for `pdfs` and `jpgs`). `NCF_sdhash` worked particularly well for small *N*s. However, more research is needed to see if this holds for larger sets, too.

flat types the removal of common features had almost no impact as most of their contents are user-generated data by design. However, `NCF_sdhash` may be superficial when processing flat types that contain layout information such as `html` to reduce the impact of template similarity.

compressed types (images) bytewise approximate matching is not the most efficient tool to detect similarities. Thus, removing features had only little effect on the results.

Further tests are required to see if these statements hold, e.g., expanding the test data by `zip`, `rtf`, `bmp`, `mp3`, and so on.

Common features were also found across different file types (see Section 4.2 and 4.3). Often these features belong to structure information, color space, font specifications, or even some well-known code used to accomplish a particular task. For our test, we found an overlap among compound files, `pdf`, and `jpg`, where all shared color information related to the embedded images/image.

Given that our test set was small, more comprehensive experiments are needed on larger sets. However, given that N is flexible, a user can define what a common block is. A more comprehensive dataset also has the advantage that it will contain *all* common features and that it can be used across multiple different scenarios / forensic cases. On the other hand, examiners could build case specific databases to filter out common blocks, e.g., the crime scene consisting of many devices belonging to one company. While it requires additional processing time, it could exclude features not found in a general set, e.g., metadata information, such as the owner, application version, or proprietary file types. Furthermore, template similarity may be reduced.

Table 16
Number of true positive matches (user-generated content) found by `sdhash` and `NCF_sdhash` for the most significant file types.

File type	Known user-generated content matches	`sdhash`	`NCF_sdhash` for *N*	
			3	5
`doc`	18	18 (100%)	16 (89%)	16 (89%)
`pdf`	12	11 (92%)	12 (100%)	12 (100%)
`text`	25	25 (100%)	23 (92%)	25 (100%)
`html`	7	7 (100%)	6 (86%)	1 (86%)

RQ2. How does ignoring common features impact the similarity detection (i.e., number of matches)?

For most tests removing the common features had a positive impact on the number of matches, e.g., reducing them from 9220 to 409 ($N = 3$ and score ≥ 1, see Table 6). It did not work equally well for the different file types but effectively for *compound types* and `pdfs`. Having less matches will be time saving from an investigator perspective. On the other hand, the quality of our results (true positives) remained similar: Having low *N*s usually found user-content generated matches although some few matches were missed.

Table 16 presents the number (and percentage) of true positives found by `sdhash` and `NCF_sdhash` (for the most promising values of $N = \{3, 5\}$) in relation to the total number of known matches (a.k.a. recall rate Davis and Goadrich, 2006). As mentioned, due to the high number of matches, a complete analysis is impossible. The values reported in the table correspond to the manually identified matches (as discussed in Section 3.2, paragraph one) which were restricted to: (1) `doc/pdf` where we analyzed all matches with score ≥ 21, (2) `text/html` where all matches with scores ≥ 5 were considered and (3) `html` where, due to the high number, we sampled 90 matches. While one may say that `NCF_sdhash` is missing matches, we argue that

- reducing manual labor significantly is most important, and
- that not all evidence has to be found during an initial run (i.e., finding one piece of evidence in 409 has a similar impact than finding three pieces in 9220.

To conclude, our results indicate that the internal file structure interferes (negatively) in the similarity identification process when the focus is user-generated content.

Another evaluation is provided by Moia et al. (2019) who analyzed how the score and recommended threshold value of `sdhash` is impacted by the removal of common features for a subset of the *t5-corpus*. By evaluating precision and recall rates of `sdhash` and `NCF_sdhash`, the authors recommend using any score > 0 given the low rates of false positives compared to `sdhash`.

RQ3. Is there a clear threshold N *for which common features are ignored in the dataset at hand?*

For our experiments, an N-value between 3 and 5 worked best in most classes. It significantly reduced the number of matches and still identified relevant matches. Unfortunately, `NCF_sdhash` missed a few matches which we considered difficult (e.g., a lot of changes performed over the whole document). This is similar to Foster (2012) who referred to blocks that repeat three or more times as common blocks.

Comparing $N = 3$ and $N = 5$: in the latter case, `NCF_sdhash` found more true positive cases but at the cost of some extra false positives. For N-values of 10 or more, we noticed an increase in the false positives where many related to template similarity.

RQ4. How does removing common features affect the runtime efficiency?

`NCF_sdhash` negatively impacts the runtime efficiency (processing time) as shown in Section 5.6 due to the db lookups. However, this difference is insignificant (for our sample set) and may be even less if using a more performant db. Furthermore, we argue that processing time is 'cheap' and it is more important to reduce the needed manual labor as discussed in the last section.

6.1. Differentiating between user-generated and other types of content

One challenge we faced in Section 5.5 was how to treat EXIF information as it can be seen as user-generated, template or application-generated content. Regardless of the category, "EXIF headers [...] can help the investigator to verify the authenticity of a picture" and is valuable evidence (Alvarez, 2004). In other words, in the case of blacklisting it should not be ignored. Every camera, user, etc. is unique, and thus there should not be too many images having the same EXIF information. Depending on N / the size of the db, they may not be considered 'common'. On the other hand, future work may consider manually modifying the db to cover these special scenarios.

More generally: before starting to remove the common features, the forensics investigator needs to define the objective of the search, finding (i) user-generated content; (ii) application-generated content; or (iii) template content. While we focused on the first (i), there may be cases where the desired matches are related to application-generated content (ii) or template content (iii). Another example is looking for template similarity, where an examiner has to find documents created by a particular company without considering their content. In such a case, every document sharing features related to the template of that company will matter.

6.2. Other applications

Apart from using the common feature database to remove undesirable features, one may use the database for other use cases. Future work is necessary to assess the significance of such approaches.

Assessing random samples quality

The database could be used for identifying objects on a device by looking for fragments of it, e.g., parsing unallocated space (Foster, 2012; Garfinkel et al., 2010; Moia and Henriques, 2016).

If found, the quality of the fragment can now be assessed. In other words, the probability a certain file was on the system is high if a distinct feature is found.

File type discovery Common features may also be used for file type identification. Given an unknown byte sequence (e.g., disk sector, object fragment), `sdhash` can be used to extract features and compare them to the db. If there are matches, we can correlate them to the file types.

6.3. Limitations

This work comes with three limitations. First, bytewise approximate matching algorithms come with some limitations, e.g., that they do not work equally well for all file types. Concerning our approach, our experiments showed that common features depend on the file type and thus do also file type dependent. Second, our experiments included a lot of manual testing and random sampling. It is possible that we misclassified matches or that we ended up with poor samples. Lastly, our results are not of general nature but only valid for our test set. For instance, for the t5-corpus, a $3 \leq N \leq 5$ works well; however other sets may require a different value. Future work is necessary to confirm if these values work for larger datasets.

7. Conclusions and future work

In this article, we analyzed the impact of common blocks (and their exclusion) for approximate matching algorithms. Therefore, we first explained what common blocks are and found that they often related to application-generated content or template similarity. Our results showed that some file types are more prone to produce common features than others and that common blocks were shared across various file types. As a second step, we removed the common blocks to see their impact on the amount and quality of similarity matches. By excluding the less important features, the number of matches was significantly reduced with an acceptable loss in the similarity detection; in some cases we obtained approximately 87% less matches compared to traditional tools.

As next steps, we want to explore different data database solutions or create a customized structure to store the common features in order to improve efficiency. Additionally, other datasets need to be analyzed to see if we can find a perfect N-value which can be universally and more file types (e.g., `zip`, `bmp`, `mp3`). Lastly, we like to study if it is possible to identify template or application-generated content similarity.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is partially supported by CAPES FORTE project (23038.007604/2014-69) and Santander International Mobility Program for Graduate Students.

References

Adobe-Systems, 2008. ISO 32000 - Document management - Portable Document 493 Format. Part 1: PDF 1.7. ISO (1st ed.).
Alvarez, P., 2004. Using extended file information (exif) file headers in digital evidence analysis. Int. J. Digital Evidence 2 (3), 1–5.
Bloom, B.H., 1970. Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13 (7), 422–426. doi:10.1145/362686.362692.
Breitinger, F., Baier, H., 2013. Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 167–182.
Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., White, D., 2014. Approximate matching: definition and terminology. NIST Special Publication 800, 168.
Breitinger, F., Roussev, V., 2014. Automated evaluation of approximate matching algorithms on real data. Digital Investig. 11, S10–S17.
Breitinger, F., Stivaktakis, G., Roussev, V., 2014. Evaluating detection error trade-offs for bytewise approximate matching algorithms. Digital Investig. 11 (2), 81–89.

Davis, J., Goadrich, M., 2006. The relationship between precision-recall and roc curves. In: Proceedings of the 23rd international conference on Machine learning. ACM, pp. 233–240.
Foster, K., 2012. Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus. Technical Report. Naval Postgraduate School Monterey (CA).
Garcia, J., 2018. Duplications and misattributions of file fragment hashes in image and compressed files. In: 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). IEEE, pp. 1–5.
Garfinkel, S., Nelson, A., White, D., Roussev, V., 2010. Using purpose-built functions and block hashes to enable small block and sub-file forensics. Digital Investig. 7, S13–S23.
Garfinkel, S.L., McCarrin, M., 2015. Hash-based carving: searching media for complete files and file fragments with sector hashing and hashdb. Digital Investig. 14, S95–S105.
Gutierrez-Villarreal, F.J., 2015. Improving sector hash carving with rule-based and entropy-based non-probative block filters. Technical Report. Naval Postgraduate School Monterey (CA).
Kameyama, A.S., Moia, V.H.G., Henriques, M.A.A., 2018. Aperfeioamento da ferramenta sdhash para identificação de artefatos similares em investigações forenses. In: Extended Anais of XVIII Brazilian Symposium on information and computational systems security. SBC, Natal-RN, Brasil, pp. 223–232.
Kim, H., Agrawal, N., Ungureanu, C., 2012. Revisiting storage for smartphones. ACM Trans. Storage (TOS) 8 (4), 14.
Kornblum, J., 2006. Identifying almost identical files using context triggered piecewise hashing. Digital Investig. 3, 91–97.
Moia, V.H.G., Breitinger, F., Henriques, M.A.A., 2019. Understanding the effects of removing common blocks on approximate matching scores under different scenarios for digital forensic investigations. In: XIX Brazilian Symposium on information and computational systems security. Brazilian Computer Society (SBC), SÃ£o Paulo-SP, Brazil, pp. 1–14.
Moia, V.H.G., Henriques, M.A.A., 2016. Sampling and similarity hashes in digital forensics: An efficient approach to find needles in a haystack. In: XVI Brazilian Symposium on information and computational systems security. Brazilian Computer Society (SBC), Niteroi-RJ, Brazil, pp. 693–702.
Noll, L. C., 2012. Fowler/noll/vo (fnv) hash. http://www.isthe.com/chongo/tech/comp/fnv/index.html, Accessed 2019 Oct 21.
Oliver, J., Cheng, C., Chen, Y., 2013. TLSH–a locality sensitive hash. In: Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth. IEEE, pp. 7–13.
Purohith, D., Mohan, J., Chidambaram, V., 2017. The dangers and complexities of sqlite benchmarking. In: Proceedings of the 8th Asia-Pacific Workshop on Systems. ACM, p. 3.
Raff, E., Nicholas, C., 2018. Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash. Digital Investig. 24, 34–49.
Rentz, D., 2007. Microsoft Compound Document File Format. OpenOffice.
Roussev, V., 2010. Data fingerprinting with similarity digests. In: IFIP International Conf. on Digital Forensics. Springer, pp. 207–226.
Roussev, V., 2011. An evaluation of forensic similarity hashes. Digital Investig. 8, 34–41.
SQLite, 2019a. Sqlite index. http://www.sqlitetutorial.net/sqlite-index/. Accessed 2019 Oct 21.
SQLite, 2019b. The sqlite query optimizer overview. https://www.sqlite.org/optoverview.html. Accessed 2019 Oct 21.

Vitor Hugo Galhardo Moia is a Ph.D student at the School of Electrical and Computer Engineering, University of Campinas, Sao Paulo, Brazil. His research interests are computer and network security, including issues related to computer forensics and applied cryptography. Moia received a master's degree in computer engineering from the same university. Contact him at vhgmoia@dca.fee.unicamp.br.

Dr. Frank Breitinger received the B.S. degree in computer science from the University of Applied Sciences in Mannheim (2009, Germany), his M.S. degree in computer science from the University of Applied Sciences Darmstadt (2011, Germany) and his Ph.D. degree in computer science from the Technical University Darmstadt (2014). He was self-employed for 5 years as well as a visiting researcher at the National Institute of Standards and Technology leading NIST SP 800-168 on Approximate Matching. From 2014 - 2019 he was an Assistant Professor University of New Haven, CT before accepting an Assistant Professor position at the Hilti Chair for Data and Application Security at the University of Liechtenstein. His research focuses on cybersecurity and digital forensics. Additional information can be found on https://www.FBreitinger.de.

Dr. Marco A. A. Henriques is an Associate Professor at the School of Electrical and Computer Engineering, University of Campinas, Sao Paulo, Brazil, where he coordinates a research group on applied security. His main research interests are in the areas of cryptography, applied security, digital identity management and cryptographic protocols. More detailed information about him can be found on http://www.dca.fee.unicamp.br/~marco.