FbHash: A new similarity hashing scheme for digital forensics

Donghoon Chang, Mohona Ghosh, Somitra Kumar Sanadhya,
Monika Singh, Douglas R. White

Indraprastha Institute of Information Technology, Delhi (IIIT-D)
National Institute of Standards and Technology (NIST)

monikas@iiitd.ac.in

July 17, 2019

Overview

Overview

Preliminaries

A major requirement of modern digital forensic investigations is an automatic filtering of the correlated/relevant data, that otherwise requires a manual examination by the investigator.

Approximate Matching

Approximate Matching algorithms is one of the techniques that reduces the amount of data an investigator has to examine manually by finding similarity at the byte level.

Approximate Matching

The approximate matching works in two phases:

Approximate Matching

The approximate matching works in two phases:

1. Digest Generation

Approximate Matching

The approximate matching works in two phases:

Existing Schemes

- ssdeep (Jesse Kornblum, 2006)
- sdhash (Vassil Roussev, 2010)
- mvHash-B(Frank Breitinger and Harald Baier 2013)
- mrsh-v2(Breitinger et al. 2013)

Active adversary attacks

Multiple previous studies have shown that these schemes do not withstand an attack by active adversaries.

An active attackers is a malicious entity, who can modify the final hash digest of file in such a way so that it can evade detection or bypass the filtering process by performing minor but intelligent modifications in the content of the file.

Active adversary attacks on existing schemes

Ssdeep:

- The paper *Security Aspects of Piecewise Hashing in Computer Forensics* by Baier et al. shows an anti-blacklisting attack on ssdeep hashes by performing intentional modification and practically proves that this approach does not withstand an active adversary against a blacklist.

Active adversary attacks on existing schemes

Sdhash:

- Paper *Security and Implementation Analysis of the Similarity Digest sdhash* shows that given a file it is easy to tamper with the file to come down to a similarity score of approximately 28, but that it is hard to overcome the matching algorithm completely.
- Paper *A Collision Attack On Sdhash Similarity Hashing* a novel approach to do maximum number of byte modification with maximal similarity score of 100. We also provided a method to do an anti-forensic attack in order to confuse or delay the investigation process.

Active adversary attacks on existing schemes

mvHash:

- Paper *Security Analysis of MVhash-B Similarity Hashing* shows that it is possible for an attacker to fool it by causing the similarity score to be close to zero even when the objects are very similar.

Example: First image from the left is original image taken from Microsoft Windows Bitmap Sample Files and the other image is the generated modified image from the proposed algorithm; mvHash-B similarity score of above images is 0

Existing Schemes

- Bottom line is none of the existing schemes are secure against active adversary attacks.

Overview

Frequency-Based Similarity Hashing

- We present a new Similarity hashing scheme called 'FbHash : Frequency-Based Similarity Hashing', which is secure against active adversary attacks.

Overview

Frequency-Based Similarity Hashing

FbHash works in the following phases:

- Similarity Digest Generation
- Similarity Score Calculation

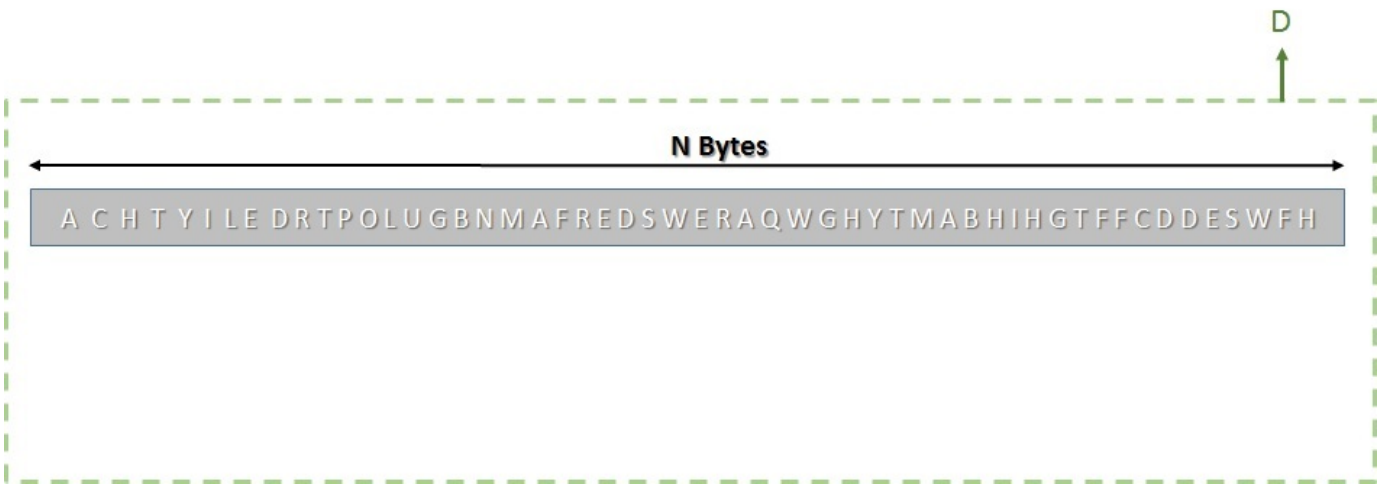
Frequency-Based Similarity Hashing

Digest generation: In order to generate similarity digest following two values needs to be calculated:

1. Chunk frequency
2. Document frequency

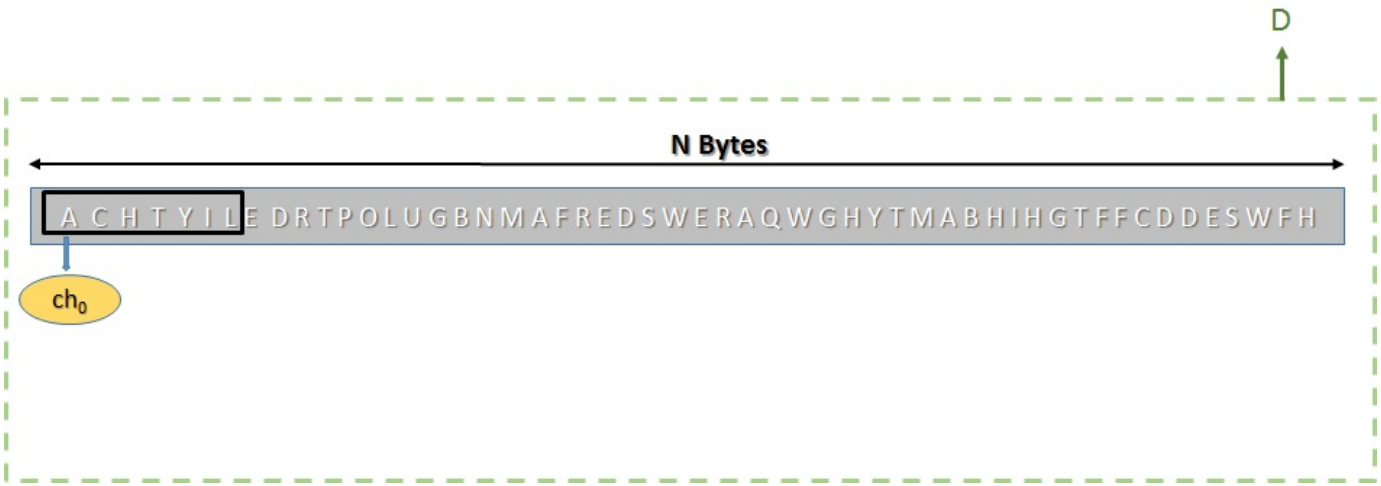
Frequency-Based Similarity Hashing

- Let D be a N byte long data object.



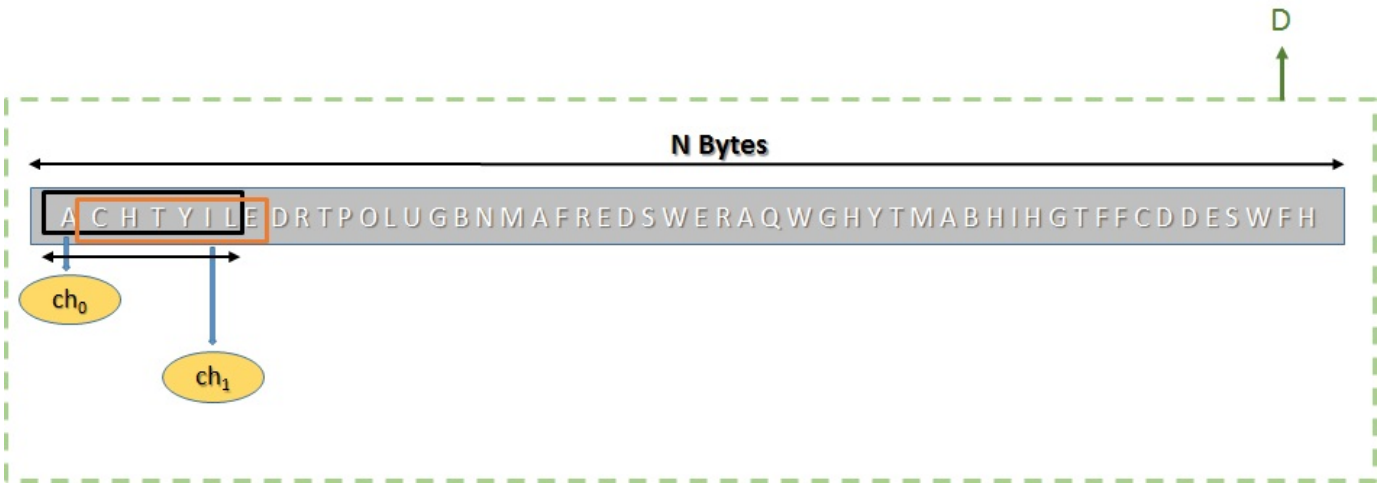
Frequency-Based Similarity Hashing

- Identify the chunks in document D (A chunk is k consecutive bytes of D:) Presently we are considering k=7.



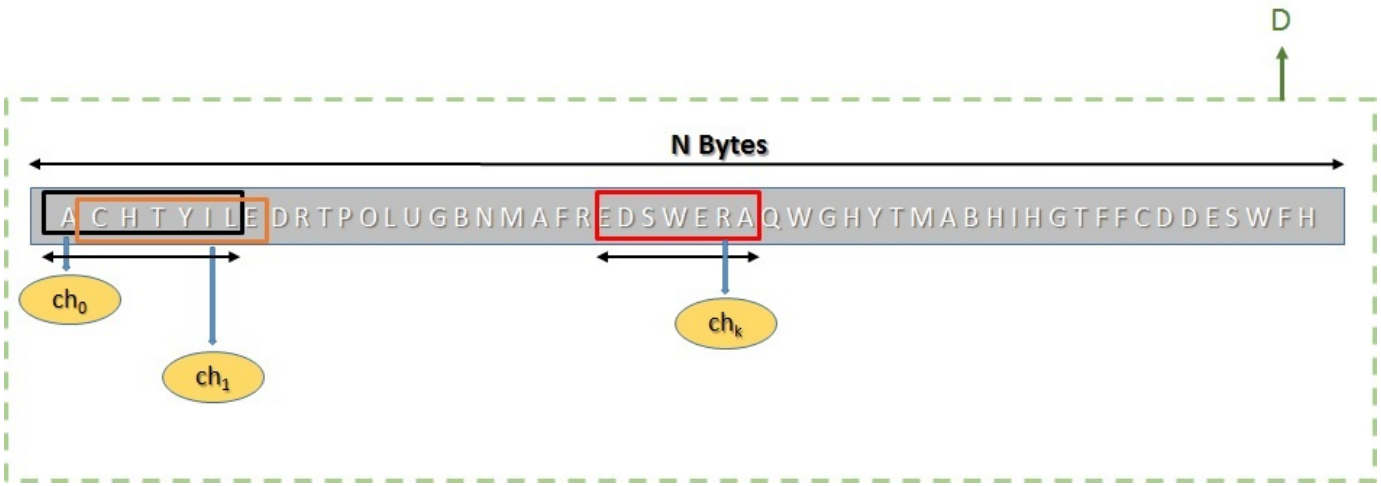
Frequency-Based Similarity Hashing

- Identify the chunks in document D (A chunk is k consecutive bytes of D:) Presently we are considering k=7.



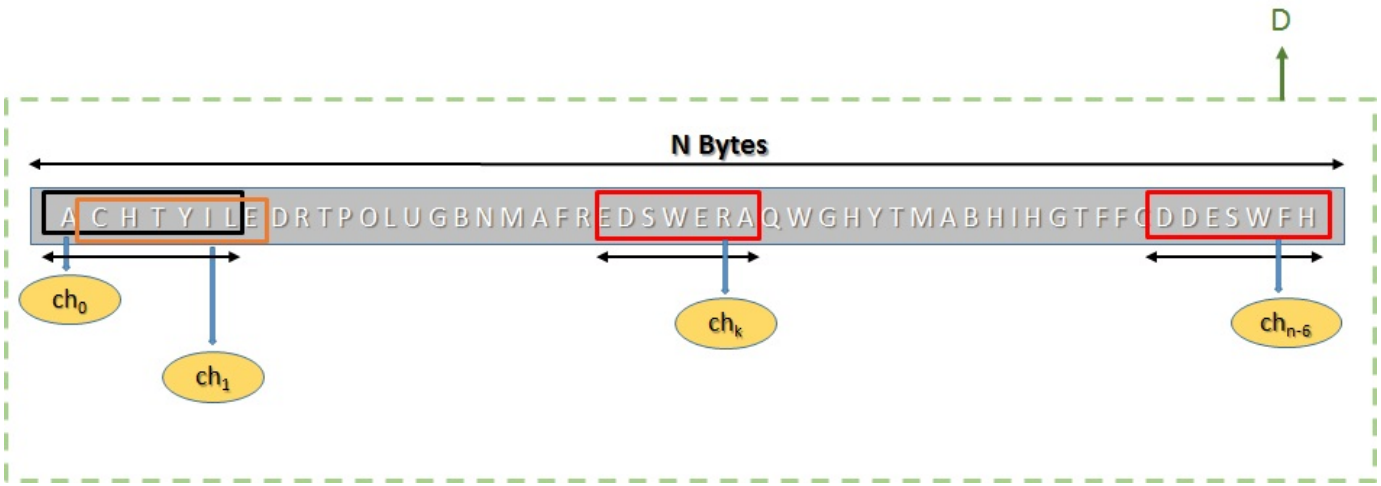
Frequency-Based Similarity Hashing

- Identify the chunks in document D (A chunk is k consecutive bytes of D:) Presently we are considering k=7.



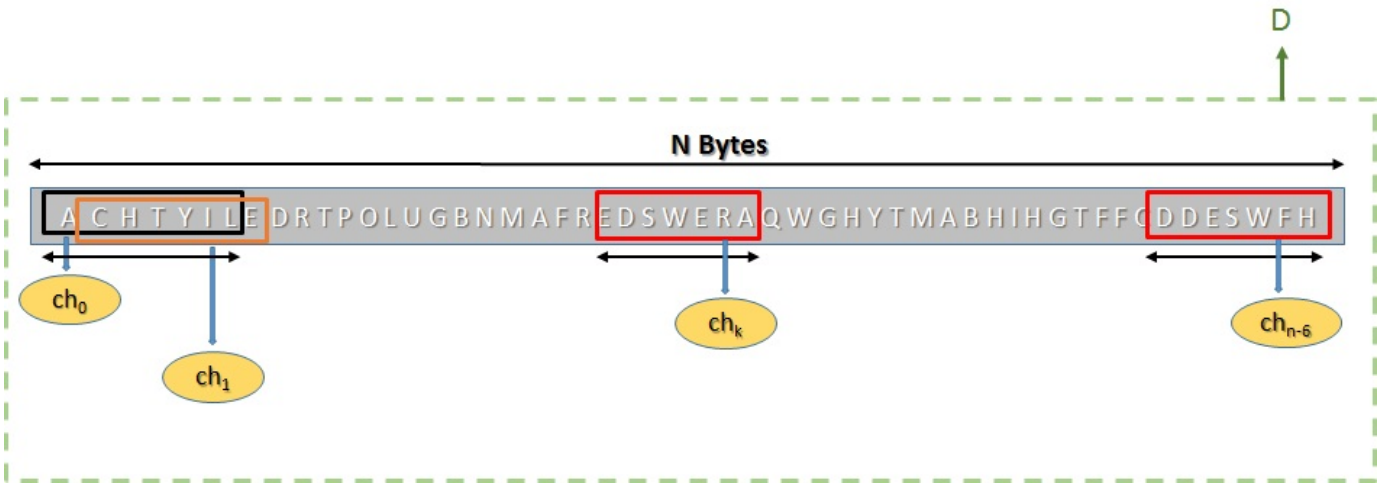
Frequency-Based Similarity Hashing

- Identify the chunks in document D (A chunk is k consecutive bytes of D:) Presently we are considering k=7.



Frequency-Based Similarity Hashing

- Rolling hash of each chunk will be calculated.



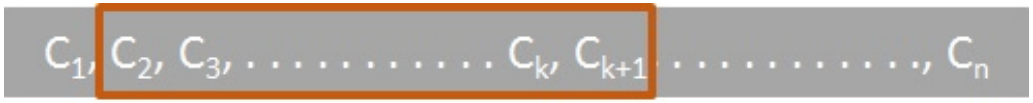
Frequency-Based Similarity Hashing: Rolling Hash

- A rolling hash is a non-cryptographic hash function where the input is hashed in a window that moves through the input.
- We are using Rabin-Karp rolling hash function, which uses a very simple function and allows very quick computation.

Frequency-Based Similarity Hashing: Rolling Hash

- $H = c_1 a^{k-1} + c_2 a^{k-2} + c_3 a^{k-3} + ... + c_k a^0$ modulus n
- Where a is a constant, k is window size, n are large prime number, and $c_1, c_2, ..., c_n$ are the input characters (ASCII values).

Frequency-Based Similarity Hashing: Rolling Hash



- The next hash value is rapidly calculated given only the old hash value, the old value removed from the window, and the new value added to the window.
- $H_{new} = a * H - c_0 a^k +$ incoming byte modulus n

Frequency-Based Similarity Hashing: Rolling Hash

- Maximum value of H can be an unsigned 64 bit number.
- Hence we need to choose window size accordingly.
- All possible values of c_i (input characters)=256 (ASCII value)
- all possible values of a (constant)=256 (size of alphabet)
- Value of k should satisfy following condition

$$2^{64} - 1 \geq 256 * 256^{k-1}$$

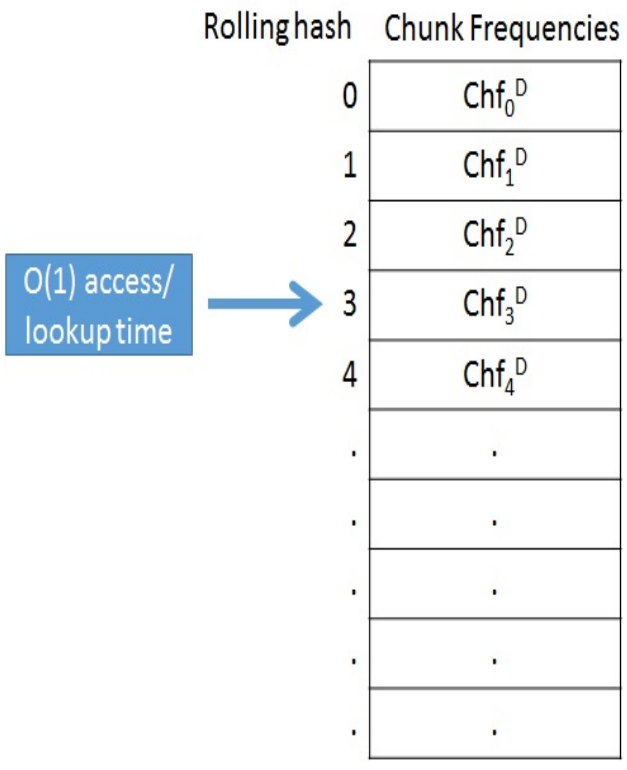
- Let k=7

$$2^{64} > 2^8 * (2^8)^6 = 2^{56}$$

Frequency-Based Similarity Hashing: Rolling Hash

Insert the rolling hash value of each chunk into a hash table. Where:

- Index of the hash table is the rolling hash value of a chunk
- Value of the hash table is the number of times a chunk appears in a document. chf_i^D represents the chunk frequency of i^{th} chunk.



Frequency-Based Similarity Hashing

- Chunk Frequency: Number of times a chunk(ch) appears in a document(D). Represented as chf_{ch}^{D}.
- Chunk Weight: Based on chunk frequency a weight will be assigned to each chunk. (Higher for higher frequency and vice-versa).
  - $\text{chunk-weight}_{ch}^{D} = 1 + \log_{10}(\text{chf}_{ch}^{D})$

Frequency-Based Similarity Hashing

Similarly we will compute the document frequency.

- Document Frequency: Number of documents that contain chunk, ch. Represented as df_{ch}.
- Document weight: Measure of chunk ch informativeness or uniqueness. Represented as idf_{ch}.
  $\text{document-weight}_{ch} = \log_{10}(\text{N}/\text{df}_{ch})$

Frequency-Based Similarity Hashing

- Chunk score:
  - W_{ch}^{D} = chunk-weight$_{ch}^{D}$ * document-weight$_{ch}$
- Now every document can be represented as vector of chunk scores.

$$\text{digest}(D) = W_{ch_0}^{D}, W_{ch_1}^{D}, W_{ch_2}^{D}, \ldots, W_{ch_{n-1}}^{D}$$

Frequency-Based Similarity Hashing

Similarity Score Calculation:

- Let D1 and D2 are documents and following is the digest of both documents:

$$\text{digest}(D_1) = W^{D_1}_{ch_0}, W^{D_1}_{ch_1}, W^{D_1}_{ch_2}, \ldots, W^{D_1}_{ch_{n-1}}$$
$$\text{digest}(D_2) = W^{D_2}_{ch_0}, W^{D_2}_{ch_1}, W^{D_2}_{ch_2,}, \ldots, W^{D_2}_{ch_{n-1},}$$

Frequency-Based Similarity Hashing

- Final similarity score between D_1 and D_2 is calculated using cosine similarity as follows:

$$Similarity(D_1, D_2) = \frac{\sum_{i=0}^{n-1} W_{ch_i}^{D_1} * W_{ch_i}^{D_2}}{\sqrt{\sum_{i=0}^{n-1} {W_{ch_i}^{D_1}}^2} * \sqrt{\sum_{i=0}^{n-1} {W_{ch_i}^{D_2}}^2}} * 100$$

- Final similarity score ranges between 0 to 100. where, 100 indicates the files are exactly the same whereas score of 0 indicates no similarity.

Why our scheme is not prone to these attacks:

Ssdeep:

- The paper Security Aspects of Piecewise Hashing in Computer Forensics by Baier and Breitinger shows and states
  - This attack was possible on ssdeep because it divides the file into big non-overlapping blocks and performs MD-5 on each chunk. One byte modification in each block will change the final hash completely.
- Our scheme: Each chunk differs by the neighboring chunk by only one byte, rest of the bytes are overlapped. Hence in order to change the final score each chunk needs to be modified. And the chuck size is smaller (7Bytes) in order to modify similarity score every 7th byte has to be modified, which is huge amount of modification.

Why our scheme is not prone to these attacks:

Ssdeep:

- The paper Security Aspects of Piecewise Hashing in Computer Forensics by Baier and Breitinger shows and states
  - This attack was possible on ssdeep because it divides the file into big non-overlapping blocks and performs MD-5 on each chunk. One byte modification in each block will change the final hash completely.
- Our scheme: Each chunk differs by the neighboring chunk by only one byte, rest of the bytes are overlapped. Hence in order to change the final score each chunk needs to be modified. And the chuck size is smaller (7Bytes) in order to modify similarity score every 7th byte has to be modified, which is huge amount of modification.

Why our scheme is not prone to these attacks:

sdhash:

- Attacks mentioned in Paper Security and Implementation Analysis of the Similarity Digest sdhash and A Collision Attack On Sdhash Similarity Hashing were possible because entire content of a file doesnt contributes to final sdhash digest generation. Only some of the selected chunks participates to final hash generation.
- Our scheme: In our scheme each and every byte of the chunks contributes to the final score and their influence on the final score depends on their importance to the document (which we decide using tf-idf calculation). In order to bring similarity score really low or close to zero almost every chunk has to be modified.

Why our scheme is not prone to these attacks:

mvHash:

- The attack mentioned in the paper was possible because during digest generation mvhash compresses the data using run-length encoding which gives the attacker freedom to bring the similarity score down with very few modification.
- Our scheme: No such compression is performed in our scheme every byte contributes to final score calculation

Overview

Comparative Analysis

We present a comparative analysis of Fbhash with the two most prominent approximate matching algorithms, (i.e., ssdeep and sdhash) on following two test-cases:

- **Fragment Identification**
  - Identify the presence of traces/fragments of a known artifact, e.g., identify the presence of a file in a network stream based on individual packets.
- **Single-common-block correlation**
  - This test aims to identify the ability of a tool to correlate the related documents, i.e., those which share a common single block of data.

Fragment Identification

Dataset Generation:

- Dataset type: Text, Docx
- Fragment size: 95%,90%,85%,80%,..5%,4%,3%,2%,1%.

Fragment Identification

Dataset Generation:

- It sequentially cuts X% of the original input length and generates the match score where X = 5 by default.
- For example file size= 100,000 bytes

Fragment Identification

Dataset Generation:

- Maximum cuts : $\lceil \frac{100}{x} \rceil - 1$
- So for a 100000 bytes long document there will be total 19 cuts of 5000 bytes.

Fragment Identification

Dataset Generation:

- In case that the algorithm still identifies similarity we continue with a further reduction in 1% steps until only 1% of the input is left.
- So the algorithm continues cutting in 1000 byte long pieces until only 1% of the input is left.

Fragment Identification

Dataset Generation:

- **Random fragment** is the first mode. The framework randomly decides whether to start cutting at the beginning or the end of an input and then continues randomly.
- **Sequential Fragment** is the second mode and only cuts blocks at the beginning of an input.

Fragment Identification

Test cases:

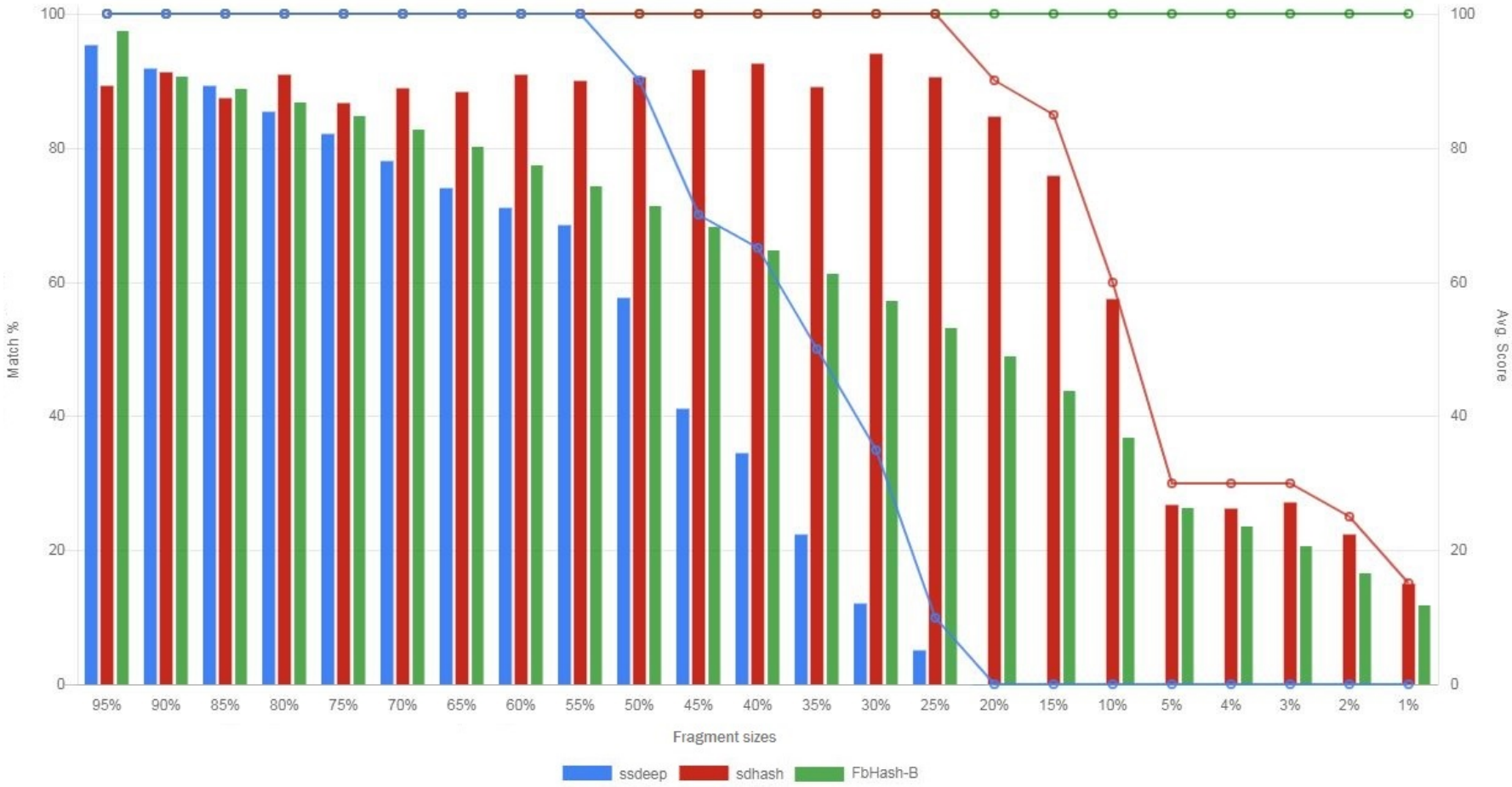
- **Fragment detection** identifies similarity tools ability to correlate a an input and a fragment.
- **Smallest Fragment Correlation** test identifies what is the smallest piece or fragment, for which the similarity tool reliably correlates the fragment to the original file?

Fragment Identification

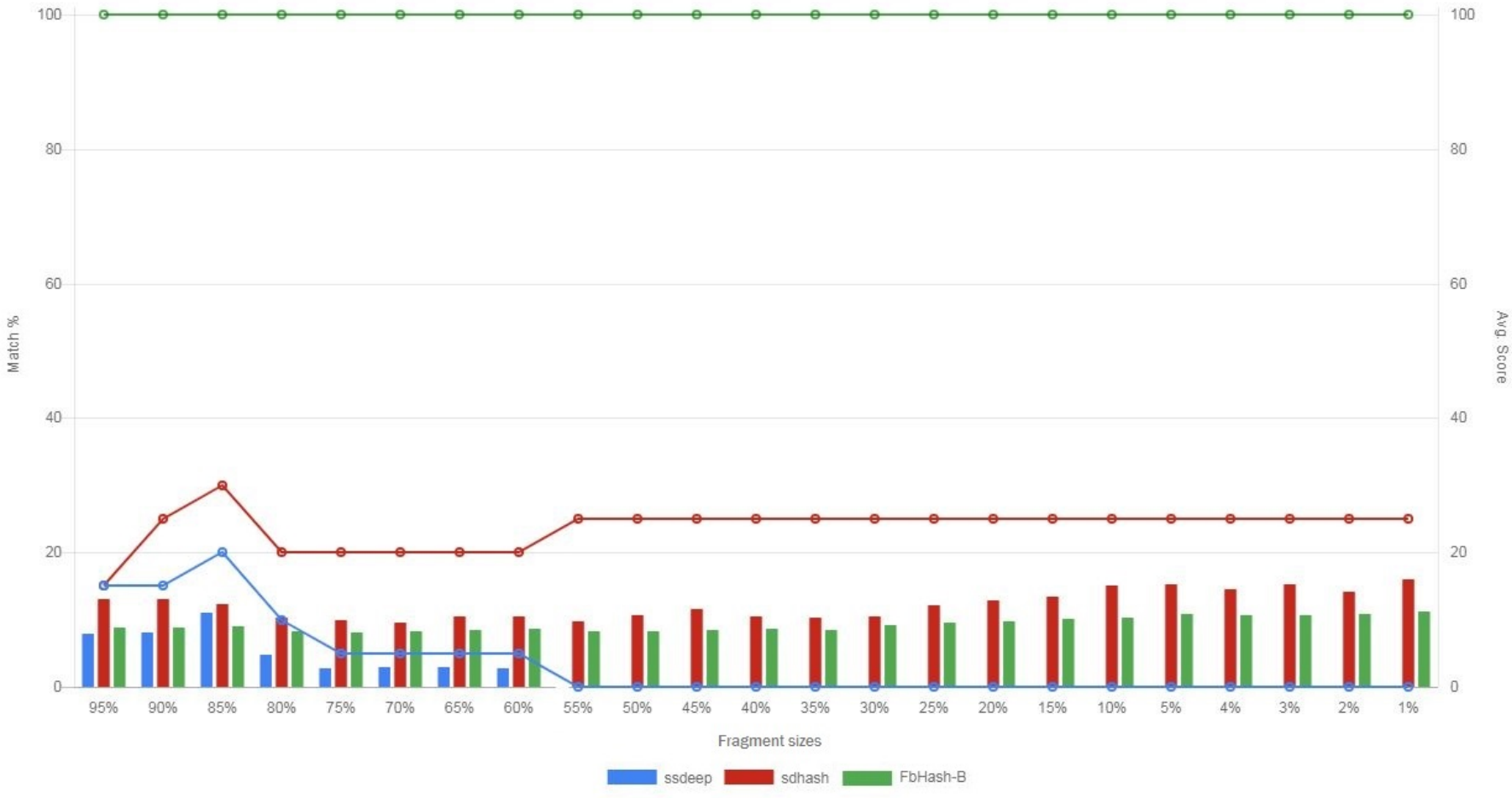
Measures:

- True Positive
- True Negative
- False Positive
- False Negative
- False positive rate (FPR)
- False negative rate (FNR)
- **Precision :** A perfect precision score of 1.0 means that every result retrieved by a search was relevant.
- **Recall :** A perfect recall score of 1.0 means that all relevant documents were retrieved by the search.
- **F-score :** A measure of a test's accuracy (best value at 1 and worst at 0)
- **MCC :** A measure of the quality of Test(best value at 1 and worst at -1)

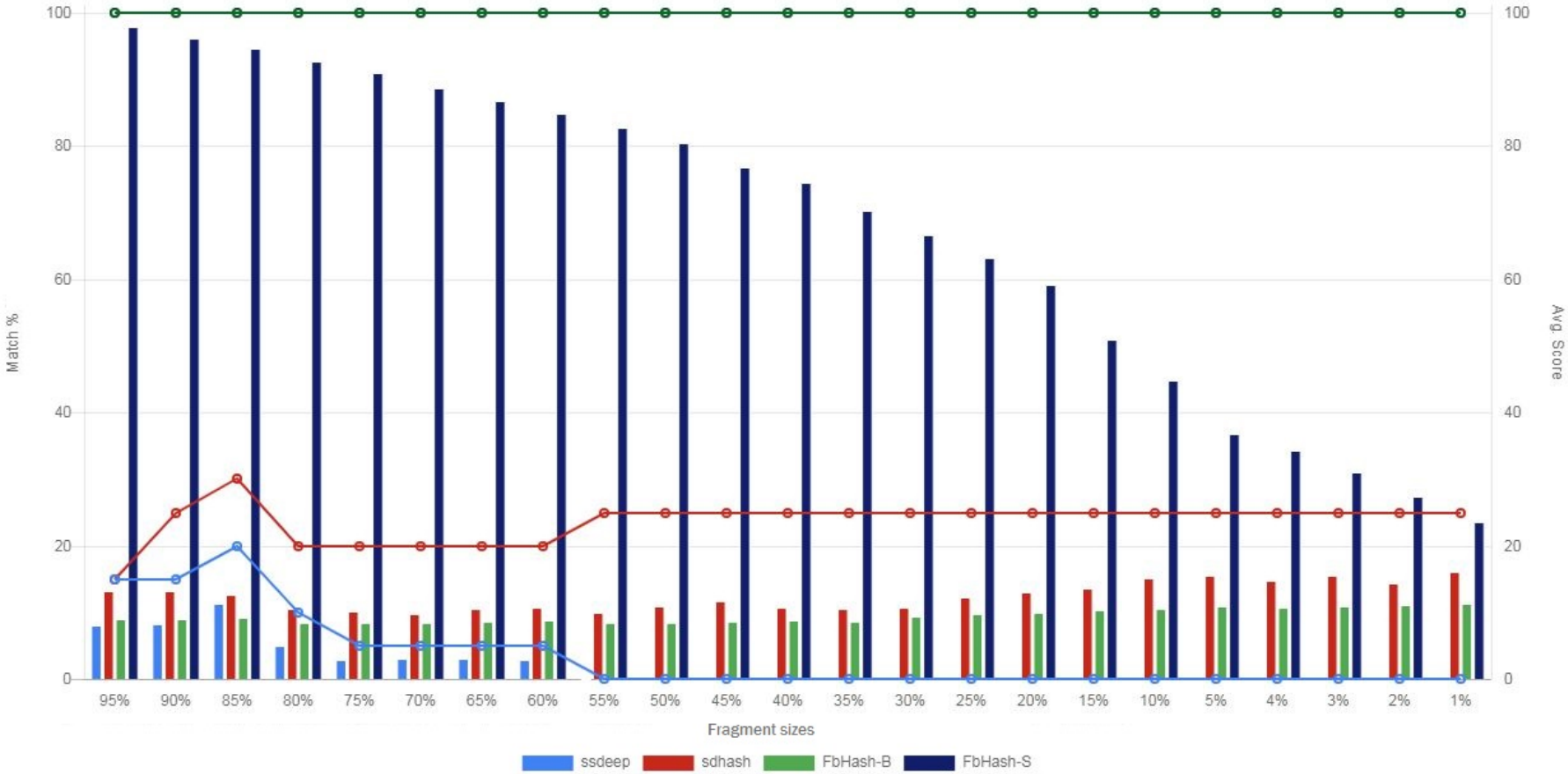
Fragment Identification:



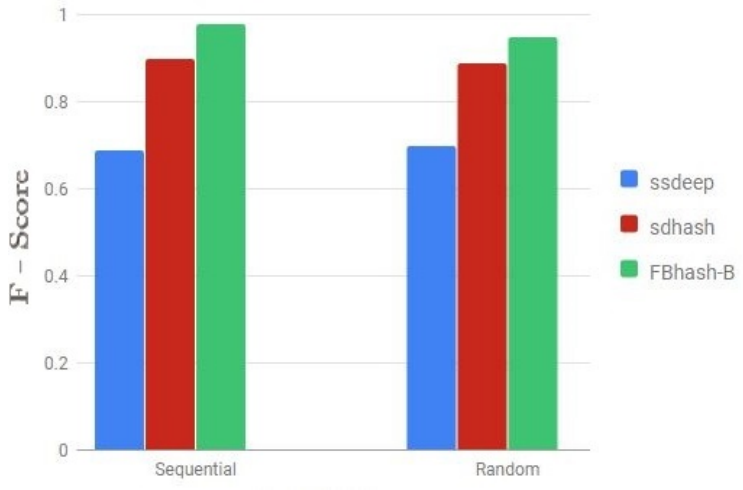
Fragment Identification:



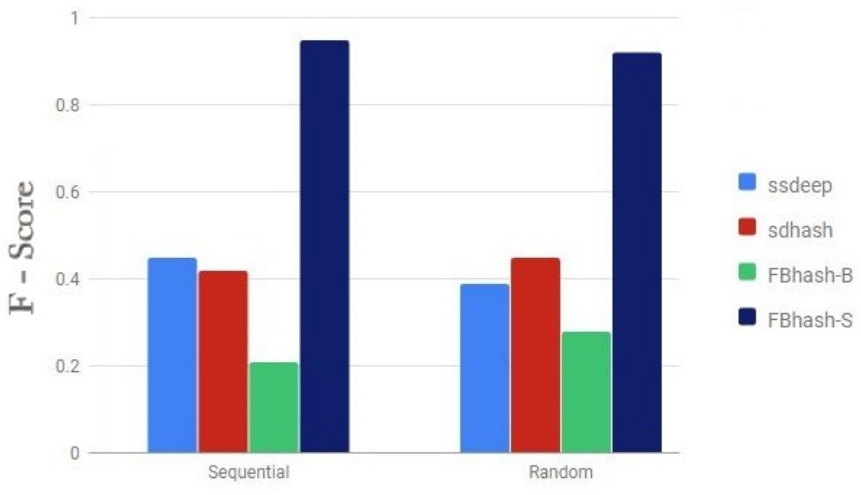
Fragment Identification:



Fragment Identification:



Fragment Identification:

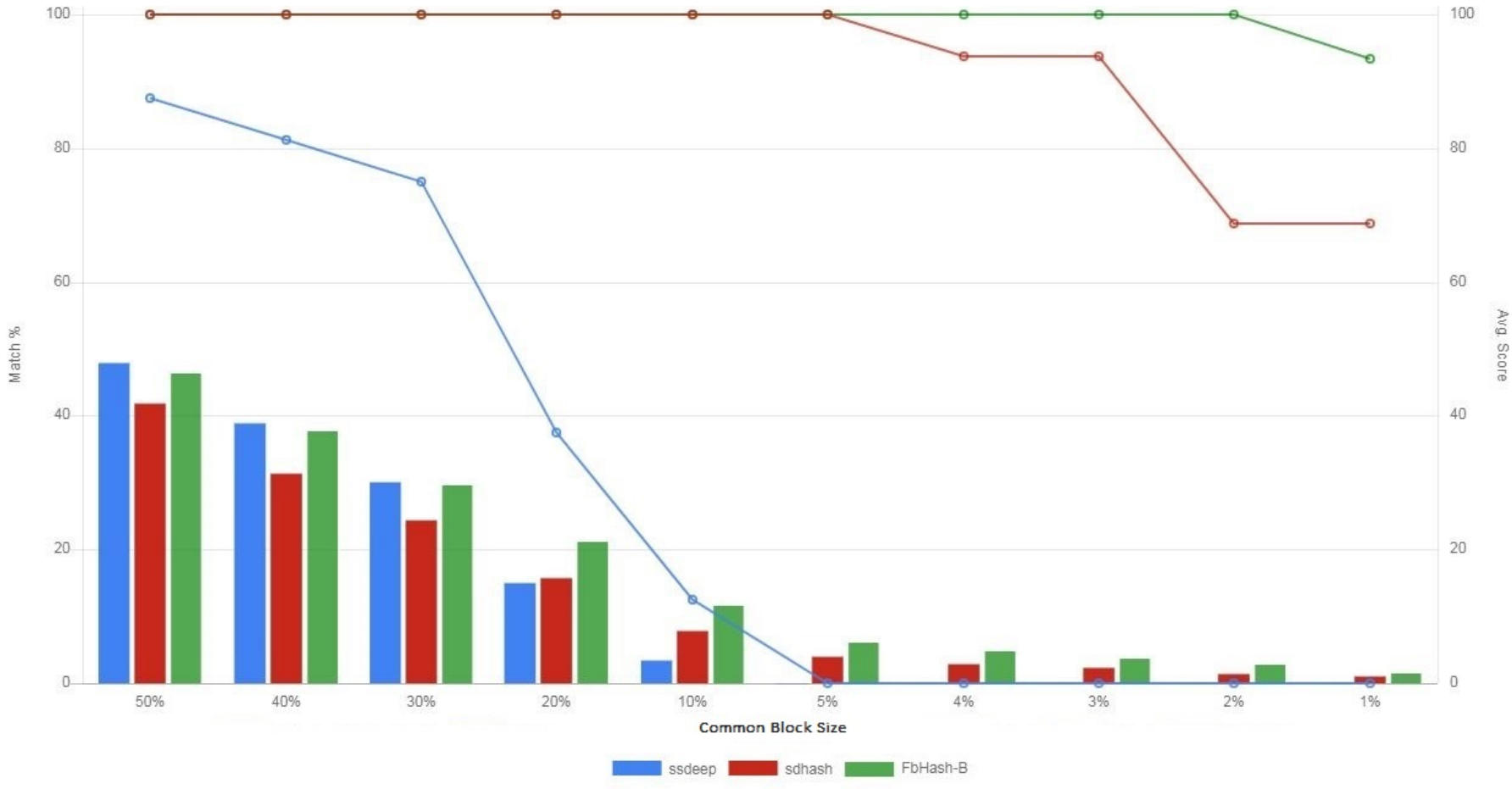


Single-common-block file correlation:

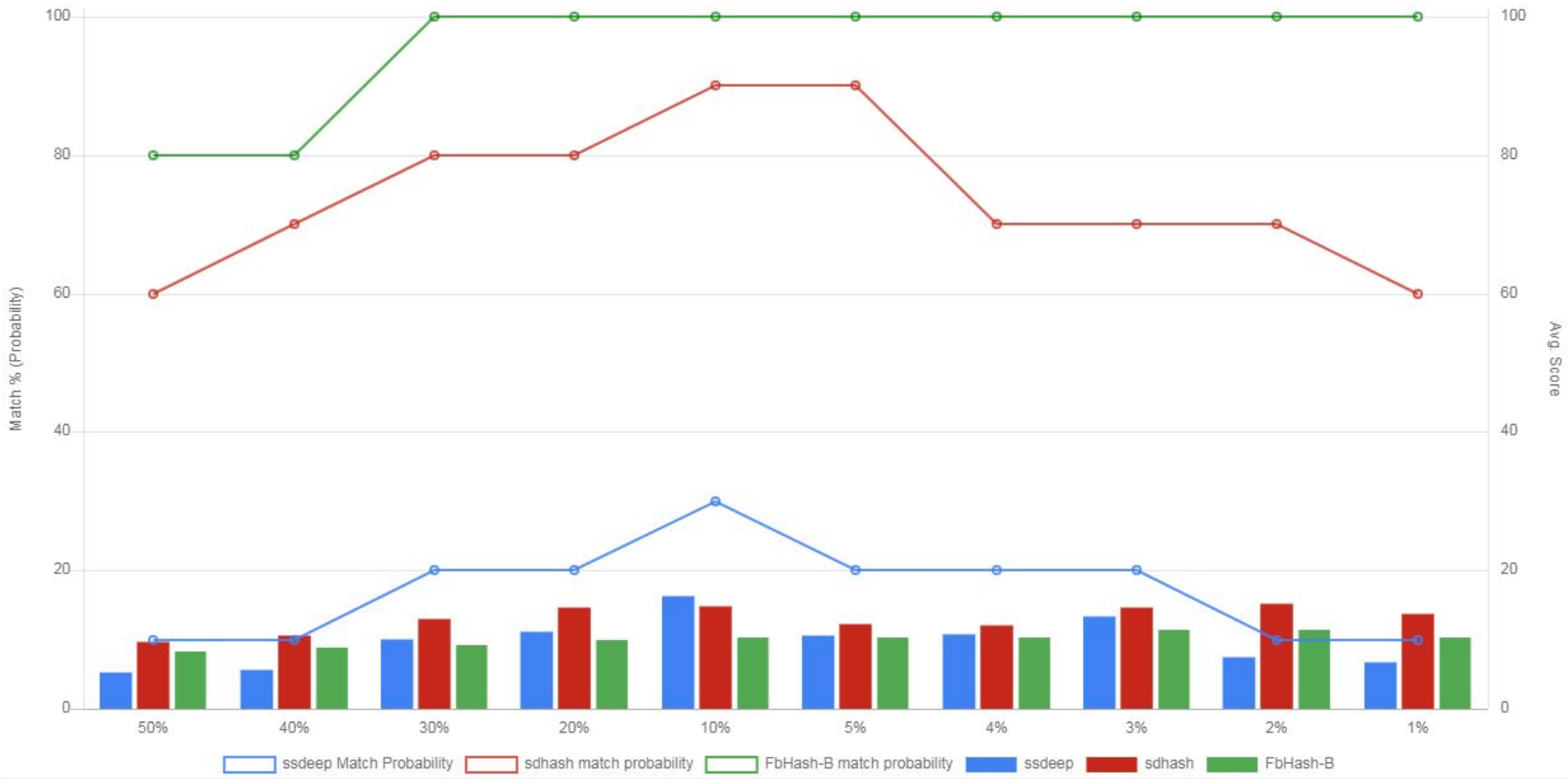
Dataset Generation the data-set is generated following the steps given below.

- 3 files of the same size are taken from the T5 corpus.
- The following 10 different sized fragments of the first file is created: 100%, 66.66%, 42.86%, 25%, 11.11%, 5.2%, 4.1%, 3.09%, 2.04%, 1.01%.
- Each fragment will be inserted in randomly chosen positions in the second and third file one by one. This will result in the creation of 10 pairs of the second and third file with shared common block of 50%, 40%, 30%, 20%, 10%, 5%, 4%, 3%, 2%, 1% respectively.
- Take another triplet of files and repeat from step 1 to step 3 for various file sizes.

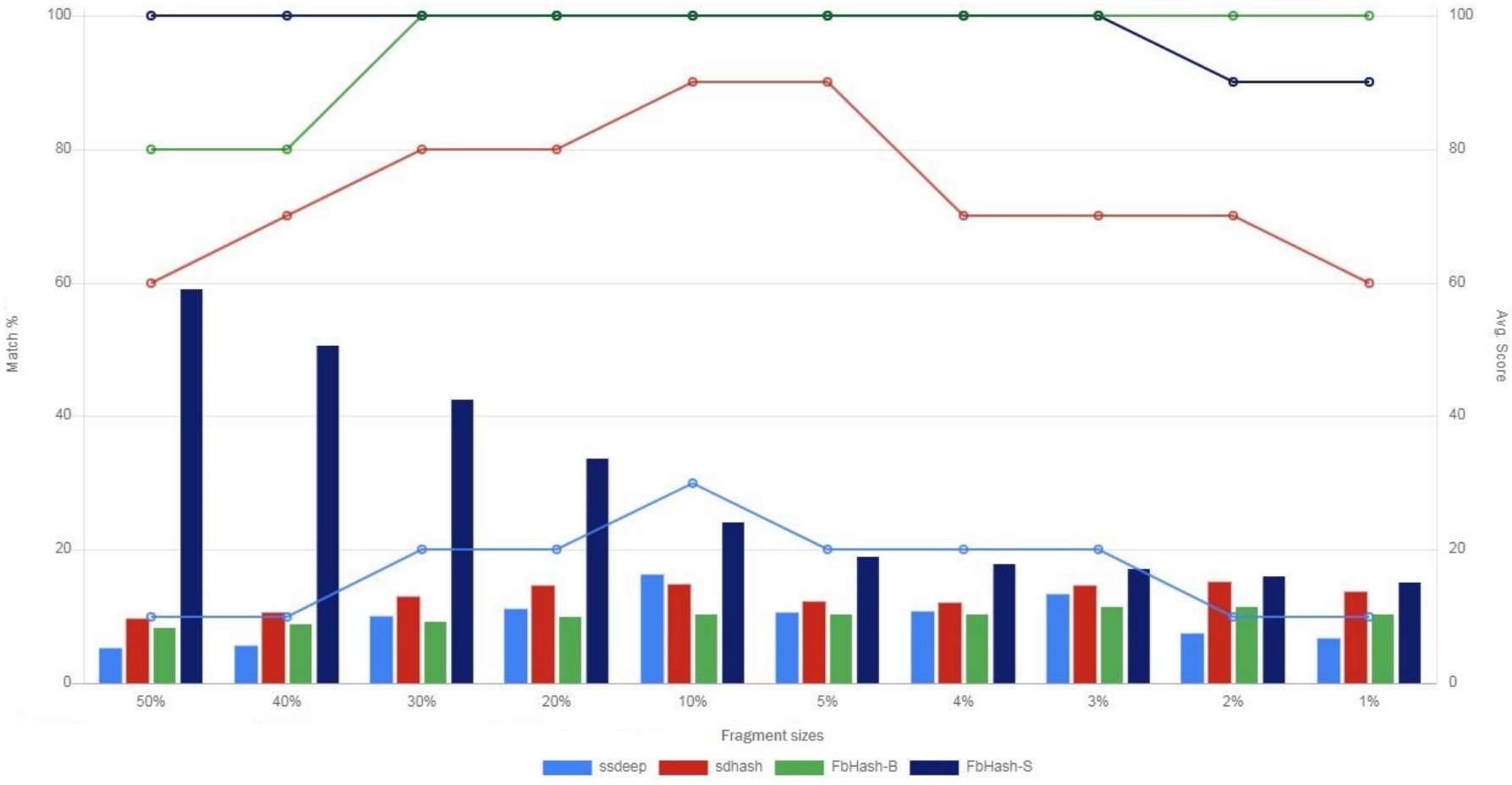
Single-common-block file correlation:



Single-common-block file correlation:



Single-common-block file correlation:



Overview

Future Work

- To analyze the runtime performance of our tool on various data-sets as a future work.
- We also plan to explore the capabilities of our tool on a few other test cases such as embedded object identification, related document detection, etc. with different types of data objects (e.g., pdf, xml etc).

Future Work

Thank You ☺