Seminar paper on Probabilistic Near-Duplicate Detection Using Simhash

Arne Beer, MN 6489196, University of Hamburg

08.07.2019

1 Introduction

In the age of the modern Internet, many services depend in large parts on crawlers and proper document detection and duplication elimination, near duplicate document detection becomes a necessity. Real time detection of which website has already been visited and whether a website is new or just has been edited are important tasks during crawling [4].

Standard hashing functions are often inefficient and operate in $O(n^2)$ space requirements for RAM and computing time [3]. At the same time, the size of available documents grow steadily. Google's website index alone has multiple hundreds of billions of web pages and over 100 Petabyte in size, according to their information website [1].

This paper attends to the paper *Seminar paper on Probabilistic Near-Duplicate Detection Using Simhash* [6]. The main challenge tackled by this paper is to find all matching pairs of fingerprints withing a certain Hamming distance h. At this time, the fasted implementation for this procedure has been *Block-Permuted Hamming Search* (BPHS), which requires RAM space at least four times the size of the whole dataset. The authors of [6] aim to design a new algorithm that allows significantly faster online and batch document comparison and furthermore reduces RAM requirements, in exchange for a small percentage of recall loss.

In the first chapters, the basics for understanding this topic will be explained. Afterwards the proposed algorithm will be looked at and the authors' findings will be discussed.

2 Conventional Hashing

Hashing is a technique, which is used to map data of an arbitrary size to a fingerprint with some fixed size. This procedure could be seen as a function $f(i) \rightarrow j$, which produces a value j from from any value i, where $j \in H$ and H is the set of values of the fixed length s with $s \in \mathbb{N}$. Well-known hash functions are, for instance, *MD5* or *SHA265*. These hashing functions are commonly used to check whether two files are absolutely identical or, for instance, to verify that a file has not been corrupted during transport. This is possible, since these hashing functions are designed to flip half of the output hash bits on average, if an input bit changes [3]. Without this property it would be easier to change the input without the hash signature being modified. This would allow malicious third parties to, for instance, change code in a binary, without users being able to detect the change with the help of this hash and would require a full byte level comparison between the original and the copied file to verify its integrity.

If, on the other hand, one's goal is to find near duplicates, which are identical for the most part, but sometimes only differ by a few bits or bytes, this hashing approach immediately becomes useless, due to this property. Due to the need for a hashing algorithm, that creates a fingerprint based on the features and structure of the input data, *Simhash* has been created.

3 Simhash

Simhash is a procedure used to create a fingerprint of a any kind of data. This fingerprint can then be used to, for instance, inspect two files for similarity.

The process for creating such a fingerprint can completely differ depending on the features in the hashed data one is interested in. In case one wants to find similar binary files, it would be reasonable to split the data into equal chunks and use these chunks as features. For websites or documents, looking at the composition and structure of text could be a viable approach to select features for hashing The original data can then be refined to a high dimensional vector of features.

The size of available features can vary significantly and is completely in the hand of the designer for each *Simhash* implementation. It's important to note, that there exists no clear guideline on which features of a data set are interesting and which features can be ignored. The performance of a *Simhash* implementation thereby also depends on the chosen features and the respective properties of the dataset. Such features can be for instance binary chunks, file extension [5], individual words, tags or URLs [6].

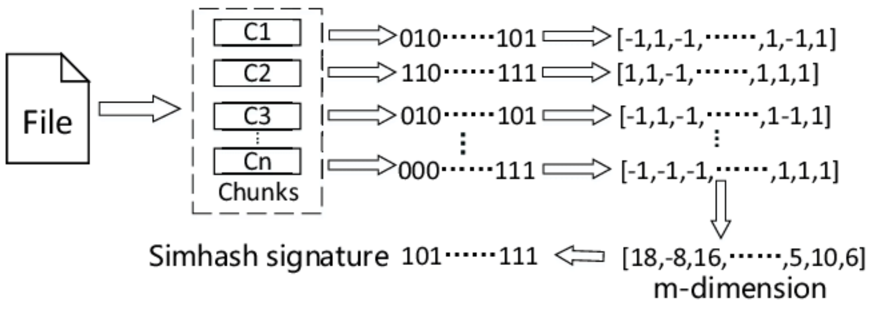

Figure 1: Visual example for calculating a *Simhash* fingerprint. A binary file split into chunks, which are then hashed. Combining all hashes results in the desired fingerprint. [7]

After determining in which way feature are extracted from the original data, each feature is hashed and mapped onto a fingerprint, which represents the constellation of features.

Algorithm 1 Simhash (u)

```
1: $W \leftarrow$ array of $b$ zeros
2: for $i \in \mathcal{F}(u)$ do                        ▷ Examine each feature
3:     $\phi_i \leftarrow$ UniformHash($i$)              ▷ Compute $b$-bit hash
4:     for $j = 1$ to $b$ do                            ▷ Iterate through each bit
5:         if $\phi_{ij} = 1$ then                      ▷ $j$-th bit of $\phi_i$
6:             $W[j] \leftarrow W[j] + w_i$              ▷ Add feature weight
7:         else
8:             $W[j] \leftarrow W[j] - w_i$              ▷ Subtract feature weight
9:         end if
10:    end for
11: end for
12: for $j = 1$ to $b$ do                               ▷ Revisit all bits
13:     if $W[j] \geq 0$ then
14:         $B[j] \leftarrow 1$                          ▷ Positive weight, set bit to 1
15:     else
16:         $B[j] \leftarrow 0$                          ▷ Negative weight, set bit to 0
17:     end if
18: end for
19: return array $B[1 \ldots b]$                        ▷ simhash
```

Algorithm 1 explaines the process of [2] for creating fingerprints. At first one iterates over the features subset $F(u)$, which is selected from the set F of all unique features with $F(u) \in F$. Each feature is then uniformly hashed to a fix size. This uniform hashing can be seen as a random projection of a arbitrarily large feature to a fixed size. The amount of ones and zeros in this hash are then counted. A one increments the counter, a zero decrements it. The result is then stored in a vector W which has the same length as $F(u)$ under the respective index of the feature.

After calculating all values for each feature, a new binary Vector B will then be created with the same Length as W. Each value $v_i \geq 0$ at index i will result in $B[i] = 1$, while each $v_i < 0$ will reslut in $B[i] = 0$.

The next challenge, which is the main topic this paper deals with, is to compare a fingerprint to all other fingerprints inside the given dataset.

3.1 Weights and Hamming Distance

3.2

3.3 Achievements of Session Juggler

4 Impact in the scientific community

5 Relevance as of 2018

References

[1] Google. *How Search organizes information*. 2019. URL: `https://www.google.com/search/howsearchworks/crawling-indexing/` (visited on 07/02/2019).

[2] Monika Henzinger. “Finding near-duplicate web pages: A large-scale evaluation of algorithms”. In: Jan. 2006, pp. 284–291. DOI: `10.1145/1148170.1148222`.

[3] A.G. Konheim. *Hashing in Computer Science: Fifty Years of Slicing and Dicing*. July 2010. Chap. 8. DOI: `10.1002/9780470630617`.

[4] Hsin-Tsang Lee et al. “IRLbot: Scaling to 6 Billion Pages and Beyond”. In: *ACM Transactions on the Web (TWEB)* 3 (Jan. 2008), p. 8. DOI: `10.1145/1541822.1541823`.

[5] Caitlin Sadowski and Greg Levin. “SimHash: Hash-based Similarity Detection”. In: Dec. 2007. DOI: `10.1.1.473.7179`.

[6] Sadhan Sood and Dmitri Loguinov. “Probabilistic Near-Duplicate Detection Using Simhash”. In: Oct. 2011, pp. 1117–1126. DOI: `10.1145/2063576.2063737`.

[7] Yongtao Zhou et al. “EPAS: A Sampling Based Similarity Identification Algorithm for the Cloud”. In: *IEEE Transactions on Cloud Computing* PP (Feb. 2016), pp. 1–1. DOI: `10.1109/TCC.2016.2527646`.