Robust Hash Algorithms for Text

M. Steinebach[1 2 3], P. Klöckner[2], N. Reimers[2], D. Wienand[2], and P. Wolf[3]

[1] Fraunhofer SIT, Rheinstrasse 75,Darmstadt, Germany
[2] CASED, Mornewegstrasse 32, Darmstadt, Germany
[3] CoSee GmbH, Rheinstrasse 75, Darmstadt, Germany

Abstract. We discuss and compare robust hash functions for natural text with respect to their performance regarding text modification and natural language watermark embedding. Our goal is to identify algorithms suitable for efficiently identifying watermarked copies of eBooks before watermark detection.

Keywords: Robust Hashing, Text Watermarking, Evaluation

1 Introduction

While multimedia content and machine to machine data have seen a strong increase in recent years, written natural text still has an important role in information distribution, storing and distribution of knowledge as well as entertainment. Books, scientific papers, patents, news articles - it is easy to list many examples important in everyday life and work. Still, concepts for reliable authentication specifically designed for natural language text are rare. Most often they are based on cryptographic hash functions, which are secure and reliable, but often require a precision of reproduction that brings challenges to efficient data handling: While in contracts every single word may be of importance, in many other documents it is rather the meaning and the flow of ideas that counts. This is especially true if digital watermarking for natural language is applied as watermarking will change the wording but not the meaning of texts, e.g. by active/passive or enumeration modulation.

In this work we discuss and compare alternatives for natural language hashing. These hashes shall feature robustness comparable to robust image or audio hashes. As long as a human observer perceives copies of a work as the same, the hash should also be identical or at least similar. Our goal is to provide a system allowing the following work flow:

- Create a robust hash H of a text T
- Create n individually watermarked copies TM of T
- Use H to identify all n copies of TM

If no hash method robust against the embedding of a watermarking, for each TM a cryptographic hash needs to be computed and stored if we want to proof TM to be a copy of T. At the same time, the only alternative to a hash is a comparison to the original copy of T. While this is acceptable with respect to computation speed and resilience to errors, here the big drawback is the need to distribute the original text. If the application is to scan the Internet for a secret document, the document often will or at least should not be available to the searching agent.

2 Motivation

As portable eBook reader become more and more common, the sale of eBooks grows. EBook revenues for 2012 are at $1.3 billion, up 46% from 2011 [1] and forecasts for 2016 range from $5 billion [2] to $10 billion [3]. Copyright holders, i.e. publishers, will face the same challenges as the music or film industry with pirated versions of their intellectual content. The illegal distribution of eBooks is comparably simple, due to their small file size which is usually about 1 megabyte. It takes only a few minutes to find free versions of all books from the *Spiegel Bestsellerliste*. On the illegal channels one can also find the scanned version of printed books.

Watermarking on content level can help to determine the leakage if an eBook is found on an illegal channel. Watermarking works in the way that it modifies the content in non-noticeable way to include a unique ID. A publisher would then be able to check the channels if his eBooks are leaked and could then identify the source of the leaked copy.

This requires of course a method to verify that a given eBook found in one of the illegal channels belongs to the publisher. Taking the content of a found eBook and comparing it on text level with a list of all owned books would be quite inefficient. Also the publisher may want to outsource checking of these illegal channels to a third party but is not willing to hand out the content of its books.

Using a hashing algorithm like SHA-1 would fail as soon as there is a minimal modification on the content. This modification could be intentional in order not to be detected. But likely it's unintentional, due to format conversion, e.g. from ePub to PDF, due to an OCR error or maybe the eBook was split into parts. As mentioned before watermarking also introduces changes into the content of the book, hence each version would have a unique SHA-1 fingerprint.

A robust hash algorithm for text documents is therefore required. It should produce the same hash value for nearly identical contents. Obviously OCR errors, small modifications and watermarks should result in the same hash value. Still defining robustness requirements can be challenging: Should a substring, e.g. the first 10 or first 100 pages of a book, produce the same hash value?

3 Related Work

The goal to create a text authentication method robust against slight modifications is not new. As an example, plagiarism recognition faces this challenge on a regular base.

3.1 Cryptographic Hashing

Hash functions allow securely computing a short digest of a long message. Mathematically speaking, a hash function is a function that maps a variable length input message to an output message digest of fixed length. Cryptographic hash functions (such as SHA-1 or RIPEMD) are by design extremely sensitive to changes in the input data: even changing one bit in the data results in a totally different hash value.

3.2 Piecewise Hashing

Piecewise hashing, also called fuzzy hashing, combines cryptographic hashing and data segmentation. One of best know examples is Ssdeep that implements an algorithm called context triggered piecewise hashing (CTPH) presented by Kornblum in 2006 [4]. It divides a byte sequence into chunks and hashes each chunk separately using the Fowler algorithm. To represent the fingerprint of a chunk, CTPH encodes the least significant 6 bits of the FNV hash as a Base64 character. All these characters are concatenated to create the fingerprint.

3.3 Robust Hashing

Perceptual hashes usually extract features from an multimedia data which are relevant to human perception and base the digest on them; thus, by design the perceptual hash of two similar objects will be similar. Perceptual hashes are sometimes also called digital fingerprints as they allow to identify content through extracted features.

3.4 Text Hashing

In the following section we describe different hash functions for natural language text. There are many applications that use cryptographic hash functions and piecewise hashing for text hashing. These approaches have one common weakness: If the document is changed by natural language watermarking, these hash functions will fail. If piecewise hashing is applied, this depends on the relation between chunk size and distance between changes caused by watermark embedding.

4 Approaches

We implemented and evaluated three algorithms: WordToBit, Broder and SimHash. The first one is based on the simple idea of hashing each word in the input text to a single bit, while the latter two are borrowed from near duplicate detection methods used in web crawling.

4.1 Word to Bit

This algorithm creates a digest by hashing each word in a given text to a single bit. To do this, we first split the text into a list of word tokens. Then we convert each word to either 0 or 1. This conversion should map the space of world uniformly at random to the space $\{0, 1\}$. We use the least significant bit of the Java built in hashCode function of the word. Other, more efficient text hash algorithms could be used. The digest is the concatenation of all those bits and its length is thus equal to the number of words in the text.

For comparison a distance measure is required. The Hamming distance is not usable as a single deleted word causes the rest of the digest to be off. The Levenshtein distance is suitable comparably slow. If the hashes are close, computing it on small parts where the two hash values differ speeds it up.

We decided to use instead a sampling approach to measure the distance of two WordToBit hash values. One hash is designated the main hash and we then try to find sub-samples of the other hash in that main hash. The motivation behind this was to be able to detect parts of a text (e.g. a chapter) in digests. The hash is split into sub-samples of e.g. 128 bits size (equal to 128 words). We then compute the Hamming distance at each position of the main hash for all sub-samples. If the distance is below a given threshold (e.g. 1 / 4 of the sub-sample size), we assume the sub-sample to match at that position. If we can find a certain number of matches (one is usually enough) we consider the whole text to match.

4.2 Broder's algorithm

Broder's algorithm, as described in [5], uses shingling to introduce a similarity measure for message digest. As mentioned before, Broder's as well as the Charikar's SimHash algorithm have been proven to be efficient to find near-duplicates in web crawling [7].

The algorithm uses m different Rabin fingerprint functions f_i, $0 \leq i < m$. The procedure starts with f_0. Each subsequence of k tokens is fingerprinted with f_0, which leads to $n - k + 1$ 64-bit values, called shingles, for f_0. The smallest of these values is the first minvalue and the first value of the hash.
The algorithm proceeds with doing the same for f_1, $f_2 \dots f_{m-1}$. Thus the algorithm results in a hash consisting of m minvalues, which leads to a hash size of $64 \cdot m$ bits. We use $k = 8, m = 84$ in most test cases. Evaluation later showed that m can be reduced to 25, which further decreases the computation time. To estimate the similarity of two texts, we determine the number of equal minvalues in their hashes and call this B-Similarity. We consider two books to be a match if the B-Similarity is at least two.

4.3 SimHash

Charikar's random projection based approach [6] is used for finding near-duplicate web pages [7]. We adopted the proposed algorithm from Charikar and introduced

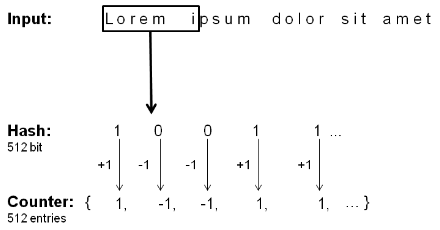

Fig. 1. SimHash mode of operation

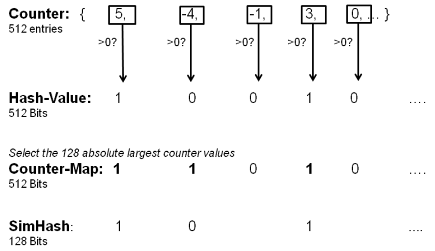
Counter: { 5, -4, -1, 3, 0, ... }
512 entries
>0? >0? >0? >0? >0?
Hash-Value: 1 0 0 1 0 ....
512 Bits
Select the 128 absolute largest counter values
Counter-Map: 1 1 0 1 0 ....
512 Bits
SimHash: 1 0 1 ....
128 Bits

Fig. 2. SimHash: Final computation step

slight changes, to yield a higher robustness. Charikar's algorithm is a fingerprinting technique that enjoys the property that fingerprints of near-duplicates differ in a small number of bit positions. We will call in the following the proposed algorithm SimHash, following the name conversion from [7].

SimHash splits works on tokens of length n. Tokens can either single characters or words. We implemented both versions, for n-grams and word sequences of length n. We decided to use the n-grams version for the ease of use and also it seemed to have better robustness properties. Each n-gram is then randomly projected to the space $\{0,1\}^k$. We use $n = 12$ and $k = 512$ and the random projection is computed by a reduced round SHA-512 implementation. As the random project does only need proper random properties, computing all rounds of SHA-512 is not necessary. Other, more efficient, pseudo-random projections could also been used.

For each n-gram we compute such a weakened SHA-512 hash value. At the same time we initialize a counter with 512 counter values. If the first bit of a SHA-512 hash equals one, we increment the first counter by one. Otherwise we decrement the counter. This is done for each bit of the hash value. Figure 1 depicts this. After hashing all n-grams and incrementing/decrementing the counter values, we convert the counter to a final hash value. In the original paper, each entry of the counter is converted to one if it is bigger than zero, else to zero. Given a random project, the expected value of each counter entry equals zero. A single changed n-gram may change the sign of a counter value, resulting to a flip in the final hash value.

To overcome this problem, we introduce a *compression step*. We select out of the 512 counter entries the 128 *most robust* entries, i.e. the entries with the largest absolute value. Changing one n-gram will not lead to a change of the sign for these counter entries. Figure 2 illustrates this final compression step.

For a given text document our SimHash algorithm returns a 512 bit large counter map and an either 128 or 512 bit SimHash value. The counter map contains a one if the counter entry belongs to the most robust entries, i.e. to one of the 128 with largest absolute value. The SimHash value can be either 128 bits, if we only convert the counters with the largest absolute values, or 512 if we simply convert all counter values.

There are several options to compute the similarity between two SimHash values. One option would be to simply compute the Hamming distance between the two compressed 128 SimHash values, ignoring the counter map. This can simply be done and the main benefit of this would be to create an easier indexing for the hash values.

For our evaluation we used a more complex comparison routine. One hash is declared as main hash. Its counter map is then used to extract from both inputs the 128 Bit SimHash value. Then the Hamming distance is computed. This introduces an asymmetric distance measure, but which could easily be made symmetric.

5 Evaluation

The evaluation of the algorithms was performed in two steps, which we call white box and black box test, respectively. White box tests should examin the properties of the present algorithms, i.e. we not only wanted to show the robustness of the algorithms but also quantify the robustness in some way. Black box tests abstract from the underlying algorithm. Here for given test scenario we were just interested in the false positive and false negative rates.

5.1 White Box Tests

To be a suitable hash algorithm, the algorithms should produce distinct hash values for distinct inputs. To test this, we extracted 1000 randomly selected articles from the German Wikipedia with a size between 9,700 and 335,000 bytes. The wiki markup was removed by WikiPrep [8] in order to gain the pure text content of the articles.

The distinction property for WordToBit is straight forward and was not further analyzed by us. As one can see in Figure 3, the mean Hamming distance of SimHash is 64, which is also the expected value for a random projection to a 128 bit space. For Broders algorithm, 998,780 out of the 999,000 possible article combinations produce a B-Similarity of zero. One pair produced a B-Similarity of 23, which is fairly high. This is due to the fact that one can find the same information or even the same paragraphs over different Wikipedia article, e.g. if a main article is split into several sub articles. In conclusion all algorithms provide sufficient distinction properties.

Revisions of Wikipedia offer a large corpus on manual modifications on text. Often a new revision changes only a typo, replaces a word or adds some information. We extract 9400 revision of various articles of the English Wikipedia to test our algorithms. We compared each revision with the previous revision of the article and computed the Levenshtein distance in order to get a measure for the performed modification. The result for SimHash and Broder can be seen in Figure 5 and Figure 6. In conclusion, both algorithm are robust to (small) human made modifications on text.

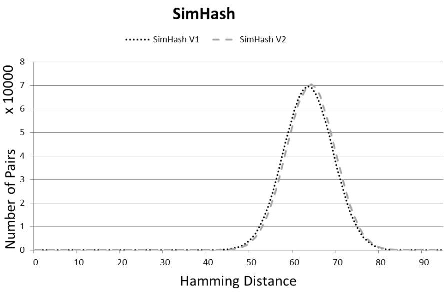

Fig. 3. Hamming distances of SimHash for 1000 randomly selected Wikipedia articles

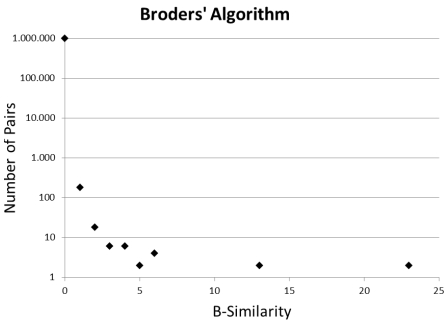

Fig. 4. Hamming distances of SimHash for 1000 randomly selected Wikipedia articles

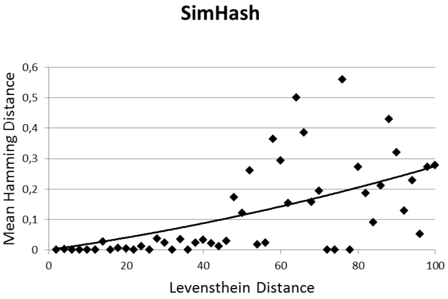

Fig. 5. SimHash revisions results

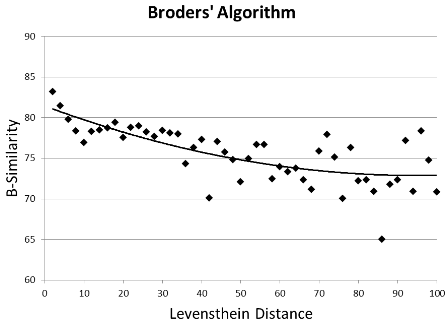

Fig. 6. Broder revisions results

As mentioned in the introduction, in some cases a book is scanned before it is distributed. Either because only the printed version is available or also to remove watermarks or circumvent DRM. We simulated typical OCR errors: I's recognized as l's, l's recognized as I's, s' recognized as f's and rn's recognized as m's. We increased the error rate from 4, 8, 25 and 10 percent in ten steps to 40, 80, 25 and 100 percent. Each bar in the following figures represents the average distance over 10 books. The distance of the SimHashes grew from zero to 10, the B-Similarity of Broders' algorithm fell reciprocally from 84 to 10.

To evaluate the performance with respect to natural language watermarking, we used eight eBooks from the current top-selling charts as covers and created marked copies of it. The watermarked versions differ from the original cover in the order of some enumerations, for example "*he was smart and cute*“ compared to "he was cute and smart“. This is one of the few accepted concepts for natural language watermarking in German language.

It turned out that using Broder with $m = 84$ leads to 3.33% of the eBooks having a B-Similarity of 82 compared to their watermarked versions, 35% a B-Similarity of 83 and 61.66% a B-Similarity of 84. SimHash even resulted in each

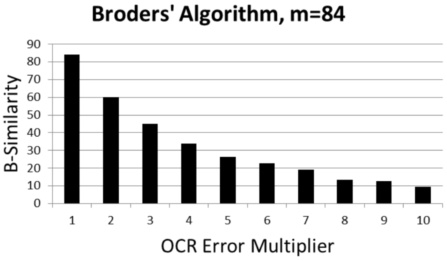

Fig. 7. Broder, OCR Simulation

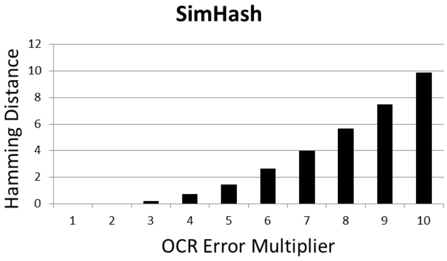

Fig. 8. SimHash, OCR Simulation

pair of hashes of the eBook and its watermarked version having a distance of zero. This shows that the given watermarking algorithm has no impact on our hash methods' ability to detect matching eBooks.

5.2 Black Box Tests

To evaluate and compare the performance of all three algorithms , we implemented an automated runner for "test tracks" taken from our corpus of texts. The test tracks are sets of matching and not matching eBook pairs assembled to represent typical use case scenarios. The first test track "Publisher" simulates the case of a publisher scanning a collection of possible copies for a single work of his. The second test track, "FBI" simulates searching a list of suspicious files for a list of known "bad" works. There is also one internal test track that contains a mixture of comparisons from the white box tests. We applied each algorithm on the given test track and logged the runtime and the number of false positives/negatives.

We tuned all three algorithms to produce zero false positives/negatives on the two main tracks. However, as our corpus of real eBooks is not nearly large enough to yield significant results regarding false positives/negatives, these tests were mainly used to assess the runtimes of the algorithms.There is a 32 bit implementation of WordToBit (using ints) and a 64 bit implementation (using longs). Benchmarks were done on a 64 bit machine. On a 32 bit machine the results for WordToBit and WordToBit64 would roughly be reversed. SimHashV1 runs on word tokens, SimHashV2 on n-grams.

The Publisher Test Track has a 1:1 ratio of hash generations to comparisons with many real books. The FBI test track contains about 1000 works and comparisons are done on the cross product of those works. WordToBit performs extraordinarily bad because its expensive operation is the comparison as opposed to the other algorithms where hashing is expensive. The internal test track again features a 1:1 hash generation:comparison ratio but with a lower number of real books, which makes WordToBit perform better than on the publisher test track.

We used black box tests to tune the parameters for WordToBit and Broder. Test runs were done on an older version of the internal test track. We ordered the results by the sum of false positives and false negatives and then by runtime.

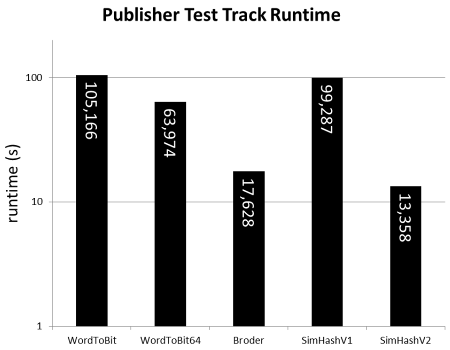

Fig. 9. Publisher Test Track Runtimes

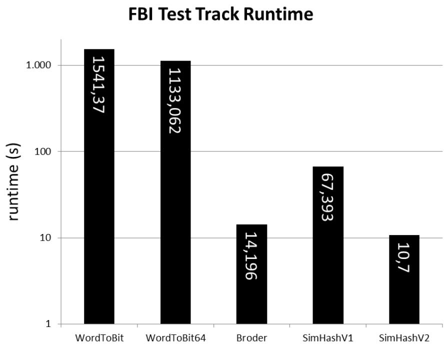

Fig. 10. FBI Test Track Runtimes

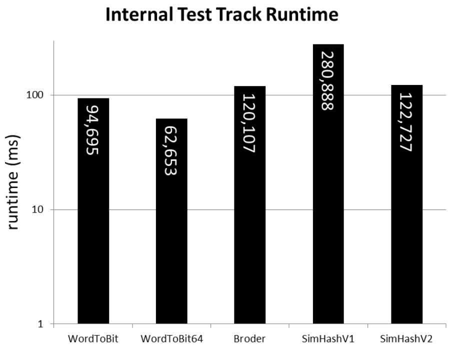

Fig. 11. Internal Test Track Runtimes

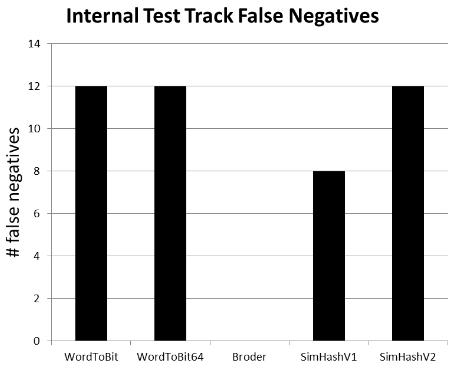

Fig. 12. Internal False Negatives

Table 1 shows the top 10 results for the parameters of WordToBit. The best combined false rates can be achieved with subsample size 3 and the distance thresholds 15 and 18. For Broder in the top 10 the false sum was always 0, therefore runtime was the only discriminator. The best runtime of 40971 ms was achieved with m=25, k=4.

6 Conclusion

As a result of our evaluation, we come to the conclusion that all the algorithms are suited for specific tasks or can be applied to satisfy certain requirements. To find chapters or in general parts of a full text we recommend the use of WordToBit. To achieve low latency one should use SimHashV2 or Broder as both are faster than WordToBit. To achieve a high precision it is advisable to use Broder as here zero false negatives could be achieved. Still, all algorithms show that it is advisable to utilize robust text hashing in error- or noise-prone environments as their robustness is an important advantage compared to common hash methods.

Table 1. Word to Bit Parameter Ranking

Rank	SubSampleSize	DistanceThreshold	Runtime	FalsePositives	FalseNegatives	False Sum
1	3	18	174904	0	3	3
2	3	15	177301	0	3	3
3	2	6	194477	0	4	4
4	2	8	196483	2	2	4
5	4	28	169801	0	5	5
6	3	12	174502	0	5	5
7	5	35	147376	0	6	6
8	4	24	168241	0	6	6
9	2	4	195907	0	6	6
10	5	30	147188	0	7	7

We show that the combined use of hashing and watermarking is applicable. Together, these tools provide a promising way to individually mark written language and identify it later on without the need of huge data bases or the leakage of the cover text. One aspect that is common in hash protocols based on similarity search is that the search and comparison part of the hash detection part must not be neglected. Depending on the application, this can become more time-demanding than the actual hash calculation.

Acknowledgement This work has been supported by the Federal Ministry of Education and Research via the project SIDIM (01IS10054A) in the funding initiative ”KMU-innovativ” for the innovative SMEs target group.

References

1. Nate Hoffelder. *AAP Reports US eBook Sales Up 46% in 2012, Now Well Over a Fifth of US Book Market.*
2. Michael Wolf. *E-book market forecast to hit $5.2B as the book industry burns.*
3. Robin Wauters. *Total Mobile eBook Sales Forecast To Reach $10B By 2016; Now Close To 1 Million Books In Kindle Store*
4. Kornblum, J. *Identifying almost identical files using context triggered piecewise hashing* Digital Investigation 3(S) (2006)
5. A. Broder, S. Glassman, M. Manasse, and G. Zweig. *Syntactic Clustering of the Web.* In 6th International World Wide Web Conference (Apr. 1997), 393-404.
6. M. Charikar. *Similarity estimation techniques from rounding algorithms.* In Proc. 34th Annual Symposium on Theory of Computing (STOC 2002), pages 380-388, 2002.
7. G. Manku, A. Jain, and A. Sarma. *Detecting near-duplicates for web crawling.* In Proceedings of the 16th international conference on World Wide Web (2007)
8. Evgeniy Gabrilovich. *Wikipedia Preprocessor (WikiPrep).* http://www.cs.technion.ac.il/ gabr/resources/code/wikiprep/