Text Origin Detection in an Efficient Way

Besnik Fetahu, Andreas Frische

February 20, 2011

Chapter 1

Introduction

The problem of detecting origin for text segments, is related closely to theft of intellectual property. There are many available solutions to this problem which are mainly categorized into broad philosophies:

- Prevention
- Detection

While the focus of this paper is related to the second category, which handles this problem in a general way, where for a framework that has training data ordered by some attributes such as date of creation, etc, one tries to find the origin of future documents comparing to the train corpus and give an answer if the content is novel or replicated.

The problem is alleviated by preprocessing of the data since from previous work referenced within the paper it gives rough estimates on the data categorization:

- 20-40 % - Exact duplicates,
- Near duplicates - almost identical, with small changes,
- Partial replication - one or more paragraphs copied, and
- Semantic duplication - same content expressed in different words.

This categorization leads to focus more on cases where one has partial replication, and thus discarding a big chunk of the data by preprocessing, this clearly has an effect on the frameworks for text origin detection, but still the amount of information is huge and one shouldn't rely on naive approaches.

This paper tackles this issues, and considers a more flexible solution with respect to space, and time performance, for which they construct a framework that uses fixed size space for saving the train data, and the output is given in a reasonable time.

On the following sections it will be shown related work and approaches used, main idea of their work, contributions, advantages and drawbacks for each of the algorithms used in their work.

Chapter 2

Related Work and Approaches

In this section is given a brief introduction to the most referenced works on this paper and give a glimpse on the ideas used, which give an intuition later on procedures used in this work.

As in this work the concept of shingles[1] is used widely in the research of text origin detection.

It allows constructing more robust structures than words itself, which also ignores irrelevant information such as formatting, capitalization, etc.

As discussed in Brin et al. (1995) shingles are also a very important aspect on constructing a framework for detection mechanism, that in a way allows to reduce the size of space used for saving the training information, and then make us of the nice properties of fingerprinting which gives you the ability that for a random bit length string you will get a string of fixed bit size. In many previous works the fingerprinting technique of Rabin M. (1981) is used, which is one of the easiest methods for implementation and as well one of the first.

Different approaches were used in Brin et al. (1995) for shingle creation, and were analyzed with respect to the space usage, giving a glimpse to the size required for saving the information with different methods of shingle creation, such as:

- One chunk equals one unit,
- One chunk equals k non-overlapping units,
- One chunk equals k units, with k-1 overlapping units.
- Non-overlapping units, determining break points by hashing units.

Important: Issues that were mentioned was security in Brin et al. (1995), where in their framework **COPS**[2] they defined a proof on what's required for the system to be secure, such as the number of characters that must be inserted,

[1] an atomic unit of k consecutive terms: (a, rose, is, a).

[2] Framework built on the research work of Brin et al. (1995) on the paper Copy Detection Mechanism for Digital Documents, which had three unit operations such as: *subset, overlap, and plagiarism.*

deleted, or modified in a given document **r** to produce a new document **r'** such that the operational units of the framework get a false value.

A visualization of space used by each approach is shown in the table below.

Strategy	Summary	Example on ABCDEF (k=3)	Space	#units	SEC <=
A	1 unit	A, B, C, D, E, F	|r|	1	|r|
B	K units, 0 over	ABC, DEF	|r| / k	K	1
C	K units, k – 1 over	ABC, BCD, CDE, DEF	|r|	K	|r| / k
D	Hashed breakpoints	AB, CDEF	|r| / k	K	|r|

Figure 2.1: Different approaches to shingle construction.

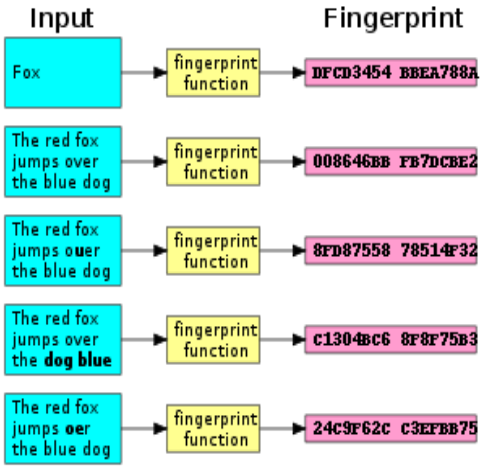

Figure 2.2: Example of fingerprinting of text messages.

They take a naive way on keeping one of the parameters secret in order to ensure that the whole framework is secure.

Chapter 3

Analysis of the Algorithm by each Phase

As mentioned previously the main two aspects to be kept into consideration were ***space-efficiency, and real-time performance.***, where the information was saved on fixed-size hash tables with a user defined allocation of space, and the hash table itself was partitioned in blocks of 64 shingles containing each.

The outline of the algorithm will be used as a guide for pointing out aspects of each phase:

- Fingerprint each document,
- Selection Algorithms - determine which shingles to save in the hash table,
- Estimation Algorithms - determine which shingles are copied from which document, and
- Eviction Algorithms - determine which shingle to remove from the hash table.

3.1 Fingerprinting of documents

In this phase as mentioned before for fingerprinting is used the method of random polynomials by Rabin M. (1981), which works on the principle that it considers each string as an n-bit message and picks a random irreducible polynomial and then the remainder of the devision between the function representing the message and the random polynomial yields the fingerprint itself.

$$f(x) = m_0 + m_1 x + ... + m_{n-1} x^{n-1} \tag{3.1}$$

$$r(x) \in GF(2) \text{ - polynomial of degree k.} \tag{3.2}$$

$$r(x) = Modulo \frac{f(x)}{p(x)} \tag{3.3}$$

\+ It's interesting properties are mainly on the fact that it has a wide use on techniques for compression, since for an arbitrary size of a message it gives fixed representation of length for each message.

- Main lack of Rabin's approach is that it's not secure in a sense that it can be easily revealed the key used to generate the fingerprints[1], thus making it possible to find the fingerprint generating polynomial and making it possible to overpass the detection mechanism.

3.2 Selection Algorithms

Selection algorithms are used for the purpose of filtering the shingles created from each document in order to get just a subset of the whole set in order to save space, and reduce overhead in computing time in latter phases. In this work they experimented with many algorithms, and compared results on latter phases to see the effects of selection algorithms in the estimation phase.

Algorithms used were:

- All,
- Every l-th (l-th),
- Modulo l (M-l),
- Winnowing w (W-w),
- Revised Hash Breaking (Hb-p),
- DCT (DCT-p), and
- Hailstorm (Hs).

All is a baseline algorithm which selects all shingles.

Every l-th a naive approach on selecting every l-th shingle from the set of all shingles.

Modulo l on this approach only the shingles whose fingerprint is divisible by a user defined value l is chosen.

Winnowing w this selection procedure developed on a research work of Schleimer S. et al. (2003), is a well defined procedure which on the first step preprocesses the data by removing irrelevant features (i.e. white space, delimiters, etc), afterwards uses the *k-gram*[2], where for efficiency they use select only a subset of shingles based on their hash values with a procedure similar to **M-l**.

\+ This has many procedure has many advantages such as, preprocessing of information, filtering of shingles where only *1/p* is retained, and it also gives guarantees that some string is detected iif it's below the **k** threshold.

[1] The algorithm requires the previous choice of a w-bit internal "key", and this guarantee holds as long as the strings r and s are chosen without knowledge of the key

[2] A k-gram is a contiguous substring of length k, taken from a string s.

- There is a trade-off on choosing the value for parameter **k** since that for bigger values we become more certain that the matches are not coincidental, thus this has the drawback that we are not able to detect substrings below this threshold.

Revised Hash Breaking Creates segments of texts where the length is determined by dividing the hash value of each word by a parameter value **p** which gives an intuition of how long the words should be.

Propose: using the original hash-breaking algorithm, where the parameter **p** is set in such a way that it discards units of length less than **p**, thus reducing noise on fingerprints.

DCT a method proposed by Seo J. and Croft W. (2008) which is similar to **Hb-p**, where the sequence of hash values created from the words are considered as a discrete time domain signal, thus applying *Discrete Cosine Transform* on those values and saving the coefficients on the fingerprints itself, this method the authors show that is more robust to noise.

Hailstorm this is a method proposed by these authors, where the basic idea as follows: first every token is fingerprinted, and afterwards the selection is based on the condition that **iif** the minimum fingerprint value occurs at the first or last position of a shingle s.

+ This method has a nice property that it gives a guarantee that any token from any query document[3] is covered by at least one k-shingle selected by the algorithm.

3.3 Eviction Algorithms

This phase is very important to the aspect of a fixed-size space solution, and the algorithms used in here try to maintain the hash table in a way that they remove information that by some conditions on each shingle its remove or retained.

In order to be able to do this, when shingles are saved in the hash table they contain information that will be used by algorithms in this phase, and the size of each shingle in bytes varies between 14-18 bytes: A usual structure of a shingle looks like this:

- Fingerprint of shingle s itself,
- Origin D_s,
- Offset D_2,
- Information about neighboring shingles in D_s,
- Information for the eviction algorithms.

[3]Query Document: The document for which we are searching if it has replicated content.

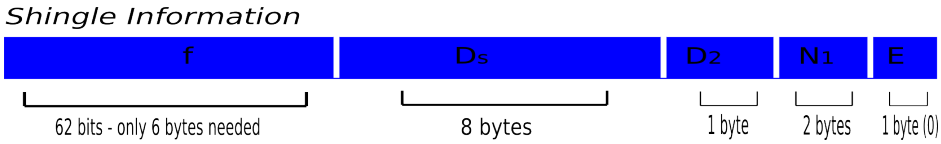

Figure 3.1: Shingle structure as it's saved in the hash table.

Main work on their research is focused on various eviction algorithms introduced before from naive approaches to those that give weights to specific shingles, which are:

- Random,
- LRU,
- Copy-Count (CC), and
- Lucky Shingle (LS).

Random is a baseline algorithm for shingle eviction, and it's approach is fairly simple and isn't promising from the eviction perspective since it evicts shingles in a random manner, and this can be the case that one could remove important shingles, therefore the result produced by this algorithm seem to be also in random values.

LRU works in a similar principle as First In - Last Out procedure, where each shingle that is inserted is placed at the end of the bucket in the hash table, and also for those searched and found shingles they are placed at the end of the bucket, whereas when needed to evict a shingle those on the top of the bucket are removed.

The approach on this method seems to perform eviction in an almost random manner since the conditions for removing a shingle are taken in a naive way.

Copy Count method has an interesting approach where it uses weights for the shingles, which is basically the number of times a shingle is searched and found in the hash table, this has the effect that shingles that are important for the framework, aren't removed, but only those with the smallest weight.

- One could argue that the score isn't sufficient enough for unseen cases of shingles where their score could be zero, and thus evicted from the hash table, but this is a trade-off of fixed-size space usage and also in general as shown later in the results for sufficiently reasonable space one shouldn't encounter this problem.

Lucky Shingle is an improvement to **CC** proposed by the authors in this paper, by making a weighted version of it, where for more important shingles are weighted more. Updates of the scores are done in the following way:

- Set lucky score to one for each shingle,

- Increment the score of the first and last shingle of a copied block by: $floor(\sqrt{b-2},$
- Increment the score of the shingle if it's first or last of its origin document,
- For every y-th shingle increment the score by one, and
- If the average score reaches some threshold, then divide the score by 2.

- This method seems reasonable at first sight where it gives more weight to important shingles i.e. if a shingle is the first or last one of the document or block, but on the other hand the drawback is that those parameters are set in a random manner, and thus it doesn't give a guarantee that they should work on any case of train corpus, since a usual approach on parameter estimation one uses cross-validation techniques from **Machine Learning**, which gives more optimized values for parameters.

3.4 Estimation Algorithms

This phase deals only with outputting an origin for a given query document, where the results on this phase are heavily influenced by previous phases on the selection and eviction procedures used beforehand. They experimented with a few algorithms, and also propose a new algorithm which will be shown below:

- No Bridging,
- Expansion,
- Bridging, and
- Bridging with Expansion.

No Bridging this is a baseline method, where the origin is outputted only for the shingles that are found in the hash table, this has also the drawback that it makes an extensive search for each shingle in the hash table which should result in a slower approach which is basically a one-on-one mapping between the shingles.

Expansion this procedure makes use of the information saved in the shingle structure, more specifically information about neighboring shingles D_2, where for each shingle **s** it checks on the hash table a shingle **s'** that matches this shingle, plus the neighboring information, and whenever a match is found that is reported as an origin to shingle **s**.

This approach on this procedure is done in a way that it removes coincidental matches between shingles, where its context is more broader and the confidence on it should be higher, which clearly makes the results more reasonable, than the output resulting from the first method.

Bridging this procedure is proposed by the authors of this paper, where they construct some properties for two shingles **s, and s'** which have to fulfill in order to output the origin the same origin for shingle **s** as the origin saved for shingle **s'**, the properties are as follows:

- offset of **s** in **D**[4] is less than the offset of **s'**,
- for both shingles **s, and s'** the same origin is saved in the hash table,
- difference of their offset in **D** is equal to the difference of their offset, stored in the hash table smaller than a given limit **T**.
- none of the shingles after **s** and before **s'** fulfill all previous properties, for which they assume that the whole block between them was copied from their origin and thus they are labeled with the same origin.

+ The extension of properties from the previous algorithm of **Expansion** where they assume for a given context one would expect that for the two selected shingles the shingles in between would have the same origin, this clearly also reduces computation overhead and the results give a more natural intuition on written texts.

Bridging with Expansion this is a mix of the two previous algorithms *Bridging and Expansion*, where for assigning an origin to a shingle one has to fulfill the properties of the *Bridging* algorithm plus two additional properties:

- first byte of the selected shingle immediately succeeding **s** in **D** matches the corresponding information in the hash table.
- first byte of the selected shingle immediately preceding **s'** in **D** matches the corresponding information in the hash table.

- As can be seen from the graph below, this algorithm expands the block of shingles that is assigned to the same origin by another shingle, which can be seen as a hopefully right labeling but this can easily lead to over-labeling as is shown in the results therefore this approach has its drawbacks since we make weak assumptions for the preceding shingles after a selected block by **Bridging** algorithm.

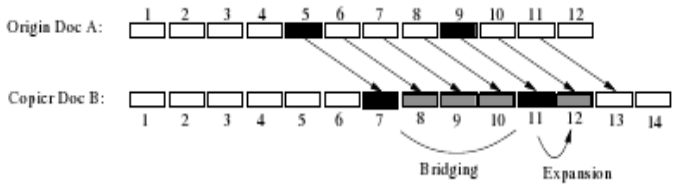

Figure 3.2: Bridging with Expansion estimation

[4]D - represents the query document for which we are trying to output the origin for each of the shingles created from it.

3.5 Results and Discussion

Performance metrics In order to determine which algorithm performs best at each phase they defined two metrics:

- **DO** - Dominant Origin which is defined as the most copied document origin **D'** from shingles of a query document **D**,
- **TF** - Token Freshness, where a token is called fresh if none of the shingles it belongs to are not found in the hash table when query document **D** is processed.

Train Information: for testing purposes they used two separate datasets: **Blogging dataset**[5], and **Swiss dataset**[6].

Experimental Set-Up since the results of each lower phase affect the higher phases, they used different experimental setups were for each phase they choose the best performing algorithms:

- Selection Phase Algorithms[7]: **All, NHs, NWs, NM3, Hb3, Dct8**.
- Eviction Phase Algorithms: All eviction algorithms were tested with selection algorithms **All, and NHs**.
- Estimation Phase Algorithms: from the selection phase they used **All, and NHs**, whereas from eviction phase they used **CC, and LS**.

Overall Estimation they conclude that the performance of the system is the best when using **NHs** with **LS** and **BE** estimation, where with the decrease of the size of the hash tables **m** one expects the performance to decrease gradually.

- Even though security wasn't their main focus, for a detection mechanism its a necessary condition to be secure where there are many cases how easily a detection mechanism could be overpassed i.e. pdf files which contain scripting language, where by a simple shuffling technique one would be unable to detect any useful information, and also for methods used in the selection phase, needs to be more done since if one is able to find out some parameters of the system then it would be easy to overpass the detection mechanism.

- This paper lacks the theoretical background, and there is no reasoning behind most of the proposed approaches they make, rather they seem to be tunned to their specific problem, which at first sight doesn't make it trustable if one would decide to use their approaches on random datasets.

+ As a very positive aspect regarding this research work, could be said is it's fixed-size space solution, where no matter the size of the train information for reasonable **m=5GB**[8] the results were quite promising.

[5] A collection of 8.6 million blog pages

[6] A collection of 1.3 million pages from the domain .ch

[7] Letter N on NHs, NWs, NM3 is used to describe the construction of shingles with non-complete overlap

[8] m - the size of the hash tables

\+ The contribution of their work to the field of origin detection of text segments seems valuable with three proposed algorithms for each phase, where the results achieved were much better performing than previous solutions.

Data& Version	All	NHs	4th	NW8	NM3	Hb3	Dct8
Blogs A	81.0	**83.6**	79.5	83.4	82.7	82.6	82.2
Blogs B	87.9	**89.1**	83.6	88.7	87.6	87.6	84.3
Swiss A	80.9	81.6	76.0	81.4	**81.8**	81.1	79.3
Swiss B	90.6	**88.0**	79.2	87.6	87.7	86.7	59.7

Figure 3.3: Selection Algorithm Performance

m	All				NHs			
	R	LRU	CC	LS	R	LRU	CC	LS
Dominant Origin Metric								
5000	83.2	93.0	**96.1**	95.0	98.8	98.8	98.8	98.8
500	75.7	77.6	**80.9**	80.7	79.8	88.2	**93.1**	87.5
50	73.2	73.4	72.6	**73.5**	74.7	75.9	**78.5**	77.8
Avg	76.2	78.9	**81.1**	80.4	82.8	86.0	**88.3**	86.7
Token Freshness Metric								
5000	97.6	96.1	96.8	**98.6**	93.5	93.5	93.5	93.5
500	86.9	83.8	84.1	**89.6**	86.2	88.1	**88.8**	86.0
50	78.2	76.7	71.8	**79.8**	77.2	**77.9**	75.1	77.1
Avg	85.7	83.7	81.8	**87.7**	84.3	**85.3**	84.2	84.4
Overall Score								
	80.7	81.3	81.4	**84.1**	83.5	85.6	86.2	**85.5**

Figure 3.4: Eviction Algorithm Performance

m	NHs + CC				NHs + LS			
	NB	E	B	BE	NB	E	B	BE
Dominant Origin Metric								
5000	**98.8**	**98.8**	97.4	98.5	**98.8**	**98.8**	97.4	98.5
500	93.1	93.1	92.1	92.7	87.5	92.3	92.6	**93.7**
50	78.5	78.8	77.9	79.2	77.8	82.4	83.0	**84.3**
Avg	88.3	88.7	87.6	88.4	86.7	89.7	89.7	**91.0**
Token Freshness Metric								
5000	93.5	93.5	**93.6**	**93.6**	93.5	93.5	**93.6**	**93.6**
500	88.8	88.6	89.3	88.6	86.0	88.4	89.5	**89.6**
50	75.1	75.5	76.4	75.8	77.1	80.0	81.5	**81.6**
Avg	84.2	84.2	85.0	84.5	84.4	86.2	87.1	**87.2**
Overall Score								
	86.2	86.4	86.3	86.5	85.5	87.9	88.4	**89.1**

Figure 3.5: Estimation Algorithm Performance

m	number of extra-labeled shingles	Accuracy
5000	6694	10.5
2000	8139	30.8
1000	33003	84.8
500	122614	96.6
200	163484	98.3
100	155107	98.6
50	154093	98.8
20	133737	98.6

Figure 3.6: Extra Labels by BE

Chapter 4

Bibliography

1. Detecting the Origin of Text Segments Efficiently Abdel-Hamid O., Behzadi B., Christoph S., Henzinger M. (2009)
2. Fingerprinting by random polynomials, Rabin O. M. (1981)
3. Copy Detection Mechanism for Digital Documents, Brin S., Davis J., Garcia-Molina H. (1995)
4. Winnowing: Local Algorithms for Document Fingerprinting, Schleimer S., Wilkerson D., Aiken A. (2003)
5. Local Text Reuse Detection Seo J., Croft B. (2008)
6. Syntactic Clustering of the Web Broder A., Glassman S., Manasse S., Zweig G. (1997)