Seminar Database Systems

Syntactic Clustering of The Web

Report

Amos Madalin Neculau

1. Introduction

Due to the exponential growth of the World Wide Web, Broder et al. propose a mechanism of clustering of documents that are the same or "roughly the same" in order to solve the problem of URLs instability and the proliferation of documents that are identical or almost the same.

The mechanism proposed by Broder et al. is an alternative to Uniform Resource Names (URNs). URNs are generalized forms of Uniform Resource Locators (URLs) which do not point to a resource directly, as URLs do, but indirectly through a Name Server. As a consequence, a name server is able to "translate" the URN to the best URL based on some criteria. Moreover, the advantage of a URN over a URL is that they are location independent and can track a resource that is renamed or moves its location, the URN being able to redirect the user to the nearest mirror of the desired resource.

To get a better insight of the topic, I consider that starting with a few concept definitions will help getting a better understanding.

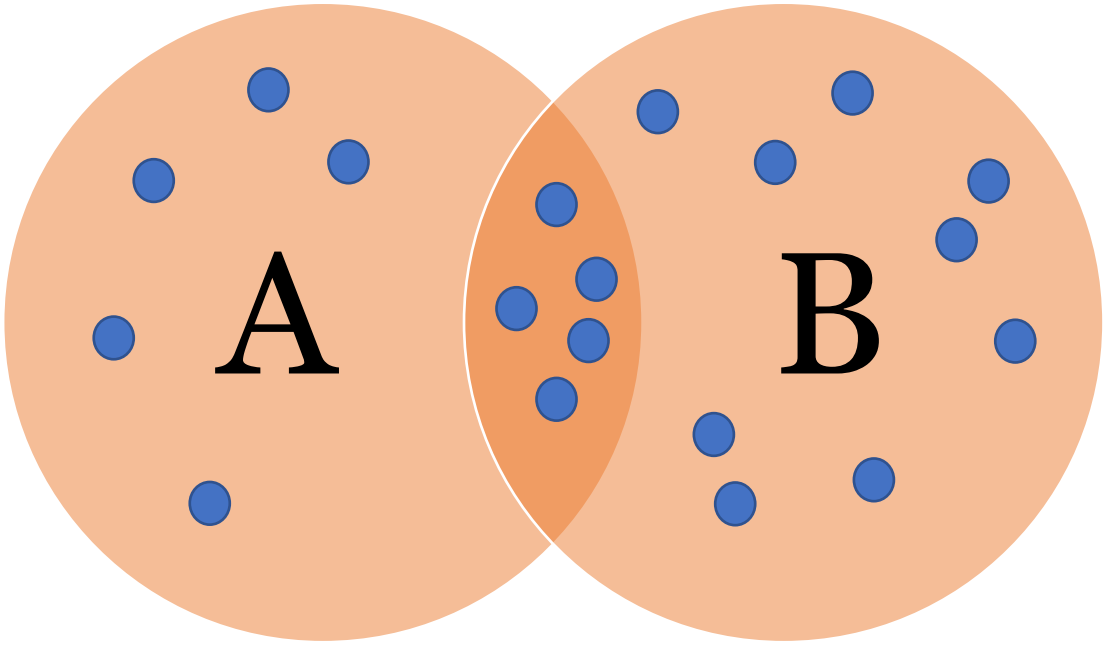
Shingle(Grams): A contiguous subsequence of tokens, e.g. words, in a given document.

W-shingle(N-Grams): Given a document D, W-shingle $S(D, w)$ is the set of all unique shingles of size w.

We can define the resemblance, also known as Jacard similarity, as follows:

Resemblance: Given two documents A and B, the resemblance $r \in \mathbb{R},\ 0 \leq r(A,B) \leq 1$ is close to 1 when the documents are "roughly the same", and close to 0 when they are different.

$$r(A,B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$, where $|A|$ is the size of the set A, and $r(A,A) = 1$.

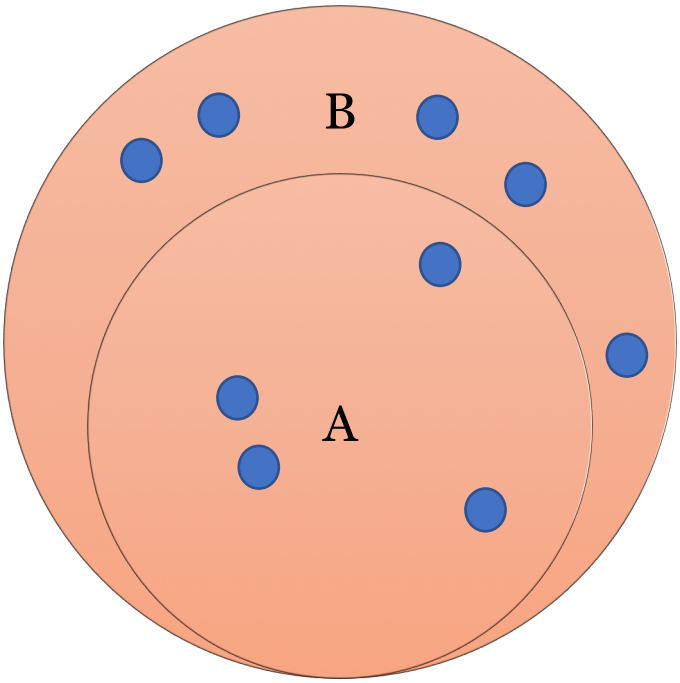


Containment: Similar to resemblance, given two documents A and B, the containment $c \in \mathbb{R},\ 0 \leq c(A,B) \leq 1$ is close to 1 when the document A is "roughly contained" within the document B, and 0 otherwise.

The containment is defined by the following equation:

$$c(A,B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$$

$$c(A,B) = 1 \iff A \subseteq B$$



Sketch: An effective method for estimating resemblance of two documents because they are easily compared, canonical representations of the documents.

In order to compute the resemblance and containment of two documents it is sufficient to keep a **sketch** of a few hundred bytes for each document, which can be efficiently computed (in time linear in the size of the document). Given two sketches, the resemblance and containment can be computed in the linear time in the size of the sketches.

Example 1

$$A = (a, rose, is, a, rose, is, a, rose)$$

$$B = (a, rose, is, a, flower, which, is, a, rose)$$

Shingle size 1 (1-Shingle)

$S(A,1) \cap S(B,1) = \{a, rose, is\}$

$S(A,1) \cup S(B,1) = \{a, rose, is, flower, which\}$

$$r_1(A,B) = \frac{|S(A,1) \cap S(B,1)|}{|S(A,1) \cup S(B,1)|} = \frac{3}{5} = 0.6$$

Shingle size 2 (2-Shingle)

$S(A,2) \cap S(B,2) = \{(a, rose), (rose, is), (is, a)\}$

$S(A,2) \cup S(B,2) = \{(a, rose), (rose, is), (is, a), (a, flower), (flower, which), (which, is)\}$

$$r_2(A,B) = \frac{|S(A,2) \cap S(B,2)|}{|S(A,2) \cup S(B,2)|} = \frac{3}{6} = 0.5$$

Shingle size 3 (3-Shingle)

$S(A,3) \cap S(B,3) = \{(a, rose, is), (rose, is, a), (is, a, rose)\}$

$S(A,3) \cup S(B,3) = \{(a, rose, is), (rose, is, a), (is, a, rose), (is, a, flower), (a, flower, which), (flower, which, is),(which, is, a)\}$

$$r_3(A,B) = \frac{|S(A,3) \cap S(B,3)|}{|S(A,3) \cup S(B,3)|} = \frac{3}{7} = 0.4285$$

Shingle Size	Resemblance
1	60%
2	50%
3	42.85%

Note: Resemblance is not transitive. Given a set of versioned documents that are "roughly the same", set size n=100. $r(doc_1, doc_2) > r(doc_1, doc_{50}) > r(doc_1, doc_{100})$, i.e. even if the versions of the papers are "roughly the same", the first version might be significantly different from version 100.

The resemblance has the additional property that $d(A,B) = 1 - r(A,B)$, is a metric (obeys the triangle inequality), which is useful for the design of algorithms intended to cluster a collection of documents into sets of closely resembling documents.

2. How to estimate the resemblance and containment

- For a fixed shingle size w, U is the set of all shingles of size w.
- W is a set, $W \subseteq U$.
- s is a parameter
- $MIN_s(W) = \begin{cases} the\ set\ of\ the\ smallest\ s\ in\ W, if\ |W| \geq s \\ W, otherwise \end{cases}$, where "smallest" refer to the numerical order on U

Fix a single size w and let U be the set of all shingles of size w. The next section discusses in detail how to estimate the resemblance and containment. The word estimate is important; this is a technique based on probability and randomness, so it's not trying to compute this metric exactly. Broder starts with two definitions. First, it is defined U, which is the set of all shingles of size w. As we'll see later the method doesn't really enumerate all possible shingles, but it's important for the descriptions below.

Second, the function $MIN_s(W)$ is defined, which the smallest s elements in the set W. This implies that U is totally ordered.

Theorem 1.

$$F(A) = MIN_s(\pi(S(A,w))$$

The paper then proceeds to the main theorem of the paper. First, they define a permutation π over the set of shingles, and then they define $F(A)$, which is the minimum s elements of the permutation of the set of shingles for document A. This function $F(A)$ is what would later be called *MinHash*. Given these preliminaries, theorem 1 states:

- The value $\frac{|MIN_s(F(A) \cup F(B)) \cap F(A) \cap F(B)|}{|MIN_s(F(A) \cup F(B))|}$ is an unbiased estimate of the resemblance of A and B.

This is the key idea in the paper. First, for completeness let's define *unbiased estimator*. This is a term from statistics that means that the expected value of this estimator, i.e. the average, will approach the true population value as the number of cases gets large. This estimator is entirely in terms of the document "sketches".

To get a better sense of this, let's look in depth on the parts of the expression above. First, we'll look at the union term which can be found in both the numerator and denominator.

$$MIN_s\big(F(A) \cup F(B)\big) = MIN_s(MIN_s(\pi(S(A,w)) \cup MIN_s(\pi(S(B,w))) =$$

$$= MIN_s(MIN_s\big(\pi(S(A,w)) \cup \pi(S(B,w))\big) =$$

$$= MIN_s\big(\pi(S(A,w)) \cup \pi(S(B,w))\big) =$$

$$= MIN_s\big(\pi(S(A,w) \cup S(B,w))\big).$$

This is a result which is used in the proof of Theorem 1, although some steps were added in order to make it clearer. This might still not be obvious, so a simple example that shows the equivalence in one case was introduced:

$$A = \{x, m, t, r\}$$

$$B = \{x, m, e, r\}$$

$$MIN_3(A) = \{m, r, t\}$$

$$MIN_3(B) = \{e, m, r\}$$

$$MIN_3(A \cup B) = MIN_3(\{x, m, t, r\} \cup \{x, m, e, r\}) = MIN_3(\{x, m, t, r, e\}) = \{e, m, r\}$$

$$MIN_3\big(MIN_3(A) \cup MIN_3(B)\big) = MIN_3(\{\mathrm{m, r, t}\} \cup \{e, m, r\}) = MIN_3(\{\mathrm{m, r, t, e}\})$$
$$= \{e, m, r\}$$

Suppose there are 10 elements in the union of $S(A,w)$ and $S(B,w)$, and 6 elements in the intersection, i.e. the resemblance is 0.6. Another way to think of this is that if you choose an element at random from the union, there's a 60% chance it's also in the intersection:

$$F(A) \cap F(B)$$

Let α be the smallest element in $\pi(S(A,w) \cup S(B,w))$. Then

$$\Pr(\alpha \in F(A) \cap F(B) = \Pr\big(\pi^{-1}(\alpha) \in S(A,w) \cap S(B,w)\big) \quad (1)$$

$$= \frac{|S(A,w) \cap S(B,w)|}{|S(A,w) \cup S(B,w)|} = r_w(A,B). \quad (2)$$

Now we look at the second part, the intersection of $F(A)$ and $F(B)$. We choose the minimum element from the previous term and call it α (actually, we can choose any element of the previous term and this still applies). What is the probability that α is in the intersection of $F(A)$ and $F(B)$?

If an item is in the intersection, then it must be in $F(A)$ **and** $F(B)$. That implies that the element before the permutation was in both $S(A,w)$ *and* $S(B,w)$. Here's the important leap in logic: **this probability is the same as the set resemblance**.

For example, as before, suppose there are 10 elements in the unions of $S(A)$ and S(B). After permuting the union, every element in the union has an equal chance of being the minimum element. If the resemblance is 0.6 there's 60% chance that the minimum element in the permuted union is also the minimum element in the permuted intersection. If it's the minimum element in the permuted intersection, it's the minimum element in both $F(A)$ and $F(B)$.

Again, the important things to note are (1) this is an *estimate* not an exact value for the resemblance, and (2) we compute the estimate entirely from the document sketches.

Taking the minimum s elements is effectively like taking a random sample of size s. The larger s is, the closer we get to the true value of the resemblance.

The system proposed by Broder et al. implement the sketches by sanitizing the web documents, take a shingle document size $w = 10$, use a 40bit fingerprint function enhanced to behave as a random permutation, and use a modulus permutation for selecting shingles with an m of 25.

Rabin Fingerprints

A fingerprint represents strings as a polynomial. It is basically a hash function but with lower collision probability. It can be computed in a rolling/windowed fashion, which works well for the shingles. The mechanism uses the Rabin fingerprint for f, both because the collision probablity is well understoond and because they can be computed efficiently for the shingles.

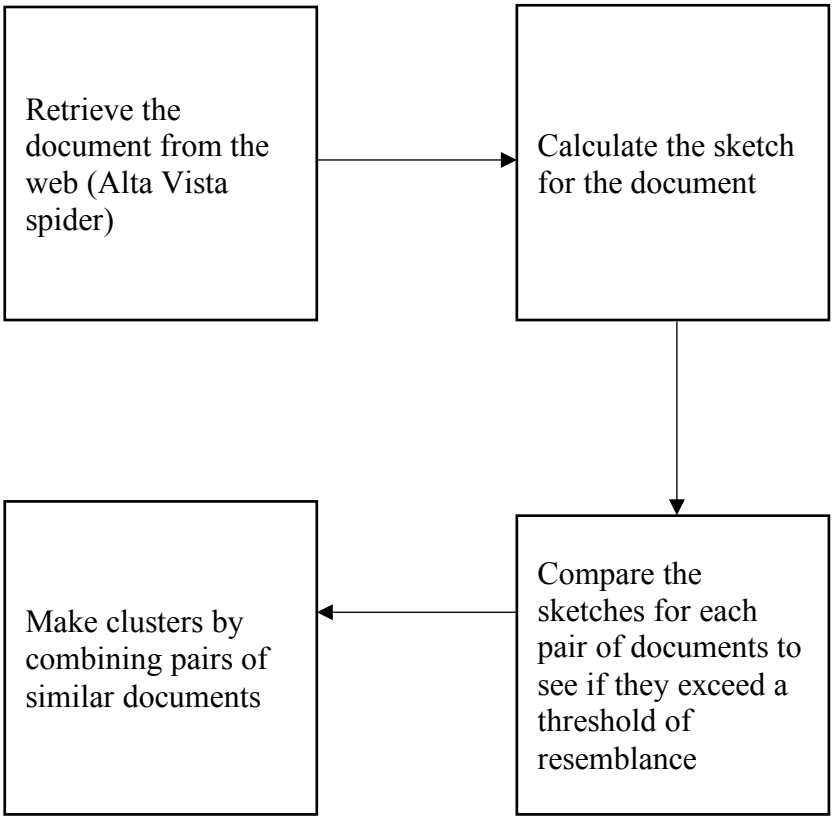
The fingerprint function uses an irreducible polynomial $p(t) \in Z_2[t]$ of an appropriate small degree k applied to a string in order to facilitate the creation of efficient and highly reliable real-time string matching algorithms.

The Permutation

To produce the sample we need a random permutation $\pi: \{0, 1, \dots, 2^l\} \rightarrow \{0, 1, \dots, 2^l\}$. The paper is a bit vague on how to generate the permutation, other than to say we need random one, or that we can basically treat the application of the Rabin fingerprint function to each shingle as the permutation. However, he does cover this in detail in a later paper on generating what he calls **min-wise** independent permutations, which means any element in the original set has an equal probability of being the minimum element in the permutation.

3. Applying the resemblance algorithm to web pages

The following diagram shows how the way the resemblance algorithm was applied on several web pages:



Despite the fact that the algorithm is simple, a trivial implementation will not inefficient. Hence the divide, compute merge approach was preferred over the trivial implementation.

Algorithm 1: Applying resemblance algorithm to the retrieved documents

Retrieve the ***data*** (***documents***) from the web

Initialize ***t*** with the threshold desired

Create an empty ***clusters*** list

For each ***document*** in ***data***:

    Calculate the ***sketch***

For each pair $< \boldsymbol{A}, \boldsymbol{B} >$ of documents:

    $Abs := Sketch_A - Sketch_B$ //Abs is the absolute value of the diference

    If $\boldsymbol{Abs} \geq \boldsymbol{t}$

        For each ***cluster*** in ***clusters***

            If $\boldsymbol{A} \notin \boldsymbol{cluster}$ AND $\boldsymbol{B} \notin \boldsymbol{cluster}$

                create new ***cluster*** and append ***A*** and ***B*** to it

                append the new ***cluster*** to ***clusters***

            Else If $\boldsymbol{A} \notin \boldsymbol{cluster}\ \boldsymbol{B} \in \boldsymbol{cluster}$

                append ***A*** to the cluster

            Else append ***B*** to the cluster

The algorithm above describes how the resemblance between two documents retrieved from the web can be calculated. Firstly, a desired threshold ***t*** must be set. This threshold will be used to decide if we add a given document to a specific cluster. The second step is to calculate the sketch for each document retrieved. Then, for each pair of documents **<*A*,*B*>** the sketches are compared to see if the threshold is reached. If so, we combine the pairs of similar documents in order to create clusters.

Algorithm 2: Divide Compute Merge (example)

Retrieve the ***data*** from the web

Initialize ***m*** with the desired size

Divide the ***data*** retrieved in pieces of size ***m*** and append it to a ***list***

A set of ***results***, initially empty

For each piece ***i*** in the ***list***

    Compute ***resemblance*** for the piece ***i***

    Merge resemblance with the results*

The Divide, Compute, Merge algorithm is used in different places in the mechanism's implementation, e.g. sorting the list composed of $< \boldsymbol{shingle\ value}, \boldsymbol{documentID} >$ pair. This is a generic algorithm and its' goal is to deal with the massive amount of data that cannot be computed

in memory. Its' scope is to take pieces of defined size m, so the computation can be done entirely in the memory. The overall performance of it is $(O(n\ log(n/m)))$. The complexity of the algorithm is due to the merge sorting (note: the list must be , which in the worst case takes $O(n\ log\ (n))$. The example stated above computes the resemblance of a document.

A summary of Divide Compute Merge algorithm:

1. Start with the pairs <shingleId, docId>.
2. Sort by shingleId.
3. In a sequential scan, generate triplets <docId1, docId2, 1> for pairs of docs that share a shingle.
4. Sort on <docId1, docId2>.
5. Merge triplets with common docIds to generate triplets of the form <docId1,docId2,count>.
6. Output document pairs with count > threshold.

Algorithm 3: The clustering algorithm

For each ***document*** in ***documents***

Calculate the ***sketch***

For ***document*** in ***documents***

For each ***shingle*** in the ***document***

If $< \boldsymbol{shingle, documentID} > \notin \boldsymbol{shingleDocIDList}$

Append $< \boldsymbol{shingle, documentID} >$ to ***shingleDocIDList***

Sort the *shingleDocIDList* using Divide Compute Merge Algorithm

For $< \boldsymbol{shingle\ value\ V_i, docID_i} >$ in ***shingleDocIdList***

If $\forall < \boldsymbol{shingle\ value\ V_j, docID_j} > \in \boldsymbol{shingleDocIdList},\ i \neq j, V_i = V_j$

For each pair $< \boldsymbol{A, B} >$ in ***documents***, **using Divide, Compute, Merge**

If $\exists < \boldsymbol{docID_A, docID_B, integer} > \boldsymbol{in\ documentsShingles}$

$< \boldsymbol{docID_A, docID_B, integer + 1} >$

Else append $< \boldsymbol{docID_A, docID_B, 1} >$ to ***documentsShingles***

Create Clusters using the resemblance algorithm to the retrieved documents (Algorithm 1)

The clustering algorithm is done in four phases. Firstly, for each document in the collection, the ***sketch*** is calculated. Then, a list of pairs of all shingles in the collection is created, along with documents they appear in. The third phase of the algorithm generates a list of all pairs of algorithms that share at least one shingle. For this, a triple is used $< \boldsymbol{docID_A, docID_B, integer} >$, where the ***integer*** is the number of shingles the documents shares. As this step requires a great amount of memory, the algorithm **Divide, Compute, Merge** is used to produce a single file of all $\boldsymbol{<docID_A, docID_B, integer>}$ triplets sorted by the first document ID. In the final phase, the clusters are created using the **Algorithm 1**.

4. Clustering overheads

4.1 Common shingles

Very common shingles cause problems during the clustering phase because the number of document ID pairs is quadratic. Furthermore, overly common shingles will have no effect on the overall resemblance of the documents or will create a false resemblance between two dissimilar documents. Hence, the very common shingles are ignored.

4.2 Identical documents

For each document, a fingerprint is generated. If there are found more than one document with the same fingerprint, all but one are eliminated from the clustering algorithm. Identical documents are found with the fingerprint of the entire original contents.

4.3 Super shingles

An alternative to estimate the resemblance of two sketches is to compute a meta-sketch. In order to compute a *super shingle,* it is required to sort the sketch's shingles, and then shingling the shingles. As a consequence, the document's meta-sketch is then determined by its set of super shingles. If two documents have even one super shingle in common, then that means their sketches have a sequence of shingles in common. In addition, the existence of a single common super shingle means it is likely that two documents resemble each other. To detect resemblance with super shingles, we only need to find a single common super shingle. Hence, super shingles are a simpler and more efficient method of computing resemblance.

However, there are negative parts on using super shingles. Some negative aspects are: super shingles are not as flexible or as accurate as computing resemblance with regular sketches. Furthermore, shingling of small documents is very inefficient. Hence, super shingles makes this problem worse. Another bad aspect is that shingles cannot detect containment.

5. Conclusion

The mechanism presented by Broder et al. allows the user to syntactically relate and cluster documents. Furthermore, a plus for this is the fact that the granularity of the shingles can be adjusted. The techniques presented can be generalized, as the resemblance and containment can be calculated on any set of objects.

6. APPENDIX:

1. How to estimate the resemblance and containment [2]:

- For a fixed shingle size w, U is the set of all shingles of size w.
- W is a set, $W \subseteq U$.
- s is a parameter

$MIN_s(W) = \begin{cases} the\ set\ of\ the\ smallest\ s\ in\ W, if\ |W| \geq s \\ W, otherwise \end{cases}$, where "smallest" refer to the numerical order on U, and define:

$$MOD_m(W) = the\ set\ of\ all\ elements\ of\ W\ that\ are\ 0\ mod\ m$$

Theorem. Let $g: U \to N$ .Let $\pi: U \to U$ be a permutation of Uchosen uniformly at random. Let $F(A) = MIN_s(\pi(S(A))$ and $V(A) = MOD_m(\pi(S(A))$. Define $F(B)$ and $V(B)$ analogously. Then

- The value $\frac{|MIN_s(F(A) \cup F(B)) \cap F(A) \cap F(B)|}{|MIN_s(F(A) \cup F(B))|}$ is an unbiased estimate of the resemblance of A and B.
- The value $\frac{|V(A) \cap V(B)|}{|V(A) \cup V(B)|}$ is an unbiased estimate of the resemblance of A and B.
- The value $\frac{|V(A) \cap V(B)|}{|V(A)|}$ is an unbiased estimate of the containment of A and B.

Proof:

$$MIN_s(F(A) \cup F(B)) = MIN_s(\pi(S(A,w)) \cup \pi(S(B,w))) = MIN_s(\pi\, S(A,w) \cup S(B,w))$$

Let α be the smallest element in $\pi(S(A,w) \cup S(B,w))$. Then

$$\Pr(\alpha \in F(A) \cap F(B) = \Pr(\pi^{-1}(\alpha) \in S(A,w) \cap S(B,w))$$

$$= \frac{|S(A,w) \cap S(B,w)|}{|S(A,w) \cup S(B,w)|} = r_w(A,B)$$

Since we can repeat this argument for every element of $MINs(\pi(S(A,w) \cup S(B,w)))$ this proves the first claim. The proof of the other two claims is straightforward.

The theorem above is roughly describing the fact that is enough to choose a random permutation and keep a sketch formed by $F(D)$ and/or $V(D)$ for a given document D in order to estimate the resemblance and containment of two documents without any need for the original files.

$F(D)$ – fixed in size, but it allows only the estimation of resemblance.

$V(D)$ – direct proportional with D's size, but it allows the estimation of both resemblance and containment.

To limit the size of $V(D)$ we can proceed as follows: for documents that have size between (say) $100 * 2^i$ and $100 * 2^{i+1}$, we store the set $V_i(D) = MOD_{2i}(g(\pi(S(D))))$. The expected size of $V_i(D)$ is always between 50 and 100. On the other hand, we can easily compute $V_{i+1}(D)$ from $V_i(D)$. (We simply keep only those elements divisible by 2i+1.) Thus, if we are given two documents, A and B, and 2i was the modulus used by the longer document, we use $V_i(A)$ and $V_i(B)$ for our estimates. The disadvantage of this approach is that the estimation of the containment of very short documents into substantially larger ones is rather error prone due to the paucity of samples.

7. References

[1] Broder, Andrei & C. Glassman, Steven & Manasse, Mark & Zweig, Geoffrey. (1997). Syntactic Clustering of the Web. Computer Networks and ISDN Systems. 29. 1157-1166. 10.1016/S0169-7552(97)00031-7.

[2] Andrei Z. Broder. On the resemblance and containment of documents.

[3] URN resource names, IETF Working Group.

[4] M.O. Rabin. Fingerprinting by random polynomials, center of research in. computer technology, Harvard University, Report TR-15-81. 1981