Applying Syntactic Similarity Algorithms for Enterprise Information Management

Ludmila Cherkasova, Kave Eshghi, Charles B. Morrey III, Joseph Tucek, Alistair Veitch

HP Laboratories
HPL-2009-90

Keyword(s):

syntActic similarity, enterprise information management, performance modeling, shingling algorithms, content-based chunking algorithms.

Abstract:

For implementing content management solutions and enabling new applications associated with data retention, regulatory compliance, and litigation issues, enterprises need to develop advanced analytics to uncover relationships among the documents, e.g., content similarity, provenance, and clustering. In this paper, we evaluate the performance of four syntactic similarity algorithms. Three algorithms are based on Broder's "shingling" technique while the fourth algorithm employs a more recent approach, "content-based chunking". For our experiments, we use a specially designed corpus of documents that includes a set of "similar" documents with a controlled number of modifications. Our performance study reveals that the similarity metric of all four algorithms is highly sensitive to settings of the algorithms' parameters: sliding window size and fingerprint sampling frequency. We identify a useful range of these parameters for achieving good practical results, and compare the performance of the four algorithms in a controlled environment. We validate our results by applying these algorithms to finding near duplicates in two large collections of HP technical support documents.

External Posting Date: May 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: May 6, 2009 [Fulltext]

To be published in Proceedings of the 15th ACM SIGKDD conference on Knowledge Discovery and Data Mining, 2009.

Applying Syntactic Similarity Algorithms for Enterprise Information Management

Ludmila Cherkasova, Kave Eshghi, Charles B. Morrey III, Joseph Tucek, Alistair Veitch

Hewlett-Packard Labs
1501 Page Mill Rd,Palo Alto, CA 94304, USA
email: {lucy.cherkasova, kave.eshghi, brad.morrey, joseph.tucek, alistair.veitch}@hp.com

ABSTRACT

For implementing content management solutions and enabling new applications associated with data retention, regulatory compliance, and litigation issues, enterprises need to develop advanced analytics to uncover relationships among the documents, e.g., content similarity, provenance, and clustering. In this paper, we evaluate the performance of four syntactic similarity algorithms. Three algorithms are based on Broder's "shingling" technique while the fourth algorithm employs a more recent approach, "content-based chunking". For our experiments, we use a specially designed corpus of documents that includes a set of "similar" documents with a controlled number of modifications. Our performance study reveals that the similarity metric of all four algorithms is highly sensitive to settings of the algorithms' parameters: sliding window size and fingerprint sampling frequency. We identify a useful range of these parameters for achieving good practical results, and compare the performance of the four algorithms in a controlled environment. We validate our results by applying these algorithms to finding near-duplicates in two large collections of HP technical support documents.

1. INTRODUCTION

The explosion of electronic documents, along with new regulations and new trends in litigation discovery, require enterprises to rethink their information management strategies. Managing the enterprise's unstructured information presents many challenges. There is increasing pressure to provide greater visibility into enterprise information for enabling different new applications, e.g., e-discovery or decreasing corporate risk from regulatory non-compliance. In particular, when compliance rules say you can (have to) delete some files, how do you identify all the users who might have a copy of these files? Often, this problem is further complicated due to the possibility that there could be many different versions of the same files (earlier or later drafts, edits of the final documents) and the same files could be (slightly) modified/altered/edited by the users over time. So, the problem transforms into one of identifying all the users who have files that are "similar" to a given one.

The focus during the e-discovery process is on identifying relevant data with high precision and recall. In many cases, once a critical document is found then the next stage is to identify and retrieve a complete set of related documents: all earlier or later versions of the same document. Often the research prototype of a successful commercial product is developed under some project "code" name. Therefore, in the enterprise archives, there could be different sets of versioned (similar) documents that reflect such project name changes and evolution of details during the project lifecycle. Failure to include a set of documents because they used a different code name could be disastrous during litigation, again making robust discovery of similar documents critical to the enterprise.

Another issue in enterprise information management is keeping document repositories with up-to-date information. Many enterprises have large collections of technical support documents, manuals, white papers and knowledge briefs on different topics. It is essential to have such knowledge stored in electronic form so that it can be searched and shared by professionals who need it. Yet inevitably these collections start suffering from different versions added over time, with older documents describing the obsolete solutions, or referencing discontinued products. Furthermore, duplicates may occur when a newly created document includes large portions from existing documents. As a part of enterprise content management, it is important to identify and filter out the documents that are largely duplicates of newer versions in order to improve the quality of the collection. Solving these information management problems requires establishing the relationships that exist within the unstructured information and techniques to exploit these relationships, e.g., content similarity, provenance, and clustering.

The problem of document similarity is not a new problem. The *question is* which of the already existing approaches and algorithms might be appropriate for solving document similarity in the enterprise environment? The problem of finding near-duplicate web pages was an area of active research in the recent past. Since near-duplicate web pages create problems for web search engines, a variety of algorithms for detecting these pages was proposed in the literature [15, 10, 19, 2, 3, 8]. A similar problem was addressed in the context of Digital Libraries [18]. A quantitative way to define that two documents are near-duplicates is to use a function that measures a degree of similarity between the objects. The similarity method is called syntactical similarity if it is entirely based on the syntactic properties of the documents rather than on their semantics. A familiar method in this group is the *diff* utility in UNIX. However, a naive approach based on pair-wise comparison is prohibitively expensive and practically infeasible for large document repositories. For designing an efficient solution, Broder *et al.* [4] used a shingling technique, which characterizes a document via a set of k-words sequences of adjacent words. Instead of using a full set of fingerprinted shingles for document comparison, an unbiased deterministic sampling technique is used to select a subset of shingles for creating a small, yet representative file signature. It is often called an approximate fingerprint-

KDD2009, June 28 - July 1, 2009, Paris, France.
.

ing (as opposed to full, exact fingerprinting).

Another popular technique for identifying similar files is called "compare-by-hash" [20] or content-based chunking [16, 9]. Originally it was proposed to take advantage of the fact that applications frequently read or write data that is identical to already existing data. To exploit these inter-file similarities, the file is divided into variable-size chunks, which are indexed by their hash value. This way, one can quickly and with high probability determine whether the contents of two files are similar by comparing only their chunk hashes and not their contents. This technique can be also classified as a syntactic similarity approach since it is entirely defined by the syntactic properties of the document.

In this paper, we compare four popular syntactic similarity algorithms: three algorithms are based on the "shingling" technique [2, 3], and the fourth algorithm is defined via "content-based chunking" [16, 9]. We define these algorithms in a unified manner using *two parameters*: a *sliding window* - that represents a contiguous sequence of bytes, and a *sampling frequency* - that defines which of these windows are included in the compact representation of a file, called a file *signature*. We observe that often these algorithms are used with very different parameter settings [4, 7, 8, 12]. The *question is* how important the choice of sliding window size and sampling frequency for the algorithm efficiency? How sensitive is the similarity metric to the different values of these parameters?

For our experiments, we use a specially designed corpus of documents that includes a set of "similar" documents with a controlled number of modifications. Our performance study reveals that a similarity metric and performance of all four algorithms is highly sensitive to different values of the sliding window size and sampling frequency parameters. Increasing the sliding window size makes the similarity algorithms much more sensitive to a number of modifications in analyzed files and significantly reduces the measured similarity metric. Decreasing sampling frequency increases variance in measured similarity metric making the outcome less robust. We identify a useful range of these parameters for achieving good practical results. We discuss potential strengths and weaknesses of each algorithm in the enterprise environment, and conclude our study by applying these algorithms to finding similar documents in two large collections of HP technical support documents. The remainder of the paper presents our results in more detail.

2. BACKGROUND AND ALGORITHMS

○ Document Similarity Definition

Two documents are considered to be similar if they are "roughly the same", i.e., they have the same content except for minor modifications, edits, or formatting. In the same way, we can define that one document is "roughly contained" within another one. To capture the intuitive definition of "roughly the same" and "roughly contained" A. Broder [2] suggested the mathematical concepts of *resemblance* and *containment* based on the *shingling* technique. Under this approach each document is represented by the set of contiguous terms (where each term is a word) or *shingles*, and then two documents are compared by the number of matching shingles. For a given document D its *w-shingling* $S^w(D)$ is defined as the subset of all unique shingles of size w contained in D (if w is fixed we use denotation $S(D)$ instead of $S^w(D)$). Then for a given shingle size, the *resemblance* or *similarity* of two documents A and B is defined as

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

The containment of A in B is defined as

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

○ Shingling-Based Approach

Rather than comparing shingles directly, it is more convenient to deal with *fingerprints* of shingles. Typically, 64-bit Rabin fingerprints [17] are used for this purpose since they have a very fast software implementation.

Furthermore, in order to reduce the number of comparisons that are needed for computing the document's similarity and containment as defined above, a few optimization techniques were proposed to approximate $sim(A,B)$ and $cont(A,B)$. They sample the set of all the shingles and build a relatively small document *signature*, and then compare these signatures. There are different ways one can sample the set of all documents shingles that result in the following three alternative similarity algorithms:

- **Algorithm** Min_n
  The Min_n method [4] selects the n numerically smallest fingerprinted shingles (or all the fingerprints if the number in the document is less than n). In such a way, each document (except very short ones) is represented by an ordered, fixed n-size signature.
- **Algorithm** Mod_n
  The Mod_n technique [4] selects all fingerprinted shingles whose value modulo n is zero. This method produces a variable length document signature (as opposed to an ordered, fixed-size signature), where the number of fingerprints in the signature is proportional to the document length.
- **Algorithm** $Sketch_n$
  Under the $Sketch_n$ approach [3] every shingle is fingerprinted with a family of n hash functions $f_1, ..., f_n$. For each f_i $(1 \leq i \leq n)$, the fingerprint of the shingle with the smallest numerical value is retained, and these values are stored in the *sketch*. This way, each document is represented by a fixed n-size feature vector. Given the feature vectors of two documents, to estimate their similarity it suffices to determine the percentage of agreeing entries in these feature vectors.

  For this approach, there is an elegant theoretical justification [3] that the expected percentage of common entries of feature vectors of two documents A and B is equal to the percentage of common shingles in all unique shingles of documents A and B.

○ Content-Based Chunking Approach

Recently, a new approach to define file similarity using content-based chunking has appeared. Content-based chunking, as introduced in [16, 15], is a way of breaking a file into a sequence of chunks so that chunk boundaries are determined by the local content of the file.

The Basic Sliding Window (BSW) algorithm [16] is one of the initial content-based chunking algorithms. This algorithm works as follows. A sliding window of fixed width w is moved across the file, and at every position k in the file, the fingerprint, F_k, of the contents of this window is computed using 64-bit Rabin's fingerprints [17]. k is a chunk boundary if $F_k \ mod \ n = 0$. The value of n determines the average size of the chunk, e.g., for $n = 100$ the algorithm produces variable-size chunks with an average chunk size of around 100 bytes. After that the "compare-by-hash" method is used to compare the chunks occurring in different files [11]. Typically, the MD5 algorithm that produces a 64 bit hash is sufficient in practice. The rationale for using the content-based chunking algorithm is that a small, local modification impacts at most two chunks surrounding the modification, and the remaining chunks would stay the same.

However, from the definition above it is apparent that it might work poorly when there are small distributed edits/modifications in a file. The issue is that a simple hash of the chunk is not resilient to small changes within the chunk like the addition or deletion of the word "the", a stem change to the term, etc. For a relatively small file, say 2 KB, even if we use ~100-byte chunking (i.e., $n = 100$) we would have ~20 chunks. If there are 10 tiny edits to the document, they potentially may impact more than 50% of the existing chunks (and their corresponding hashes). To avoid this potential drawback, we introduce a slightly modified version of the basic sliding window algorithm below.

- **Algorithm BSW_n**
  A sliding window of fixed width w is moved across the file, and at every position k in the file, the fingerprint of its content is computed. Let k be a chunk boundary (i.e., $F_k \ mod \ n = 0$). Instead of taking the hash of the entire chunk, we choose the numerically smallest fingerprint of a sliding window within this chunk. Then we compute a hash of this randomly chosen window within the chunk. Intuitively, this approach would permit small edits within the chunks to have less impact on the similarity computation.

  This method produces a variable length document signature, where the number of fingerprints in the signature is proportional to the document length.

Summary: While all the four algorithms are applicable for solving a similarity problem, not all the algorithms can approximate the containment of two documents. When document B is contained in A (say, B is the first section of a 4-section document A) then the n smallest fingerprints over sub-document B (algorithm Min_n) might have nothing in common with the n smallest fingerprints of the entire document A. The same applies to the $Sketch_n$ algorithm.

Table 1 summarizes the properties of the four algorithms.

	Mod_n	Min_n	$Sketch_n$	BSW_n
File signature size	variable size	fixed n-size	fixed n-size	variable size
Similarity	yes	yes	yes	yes
Containment	yes	no	no	yes

Table 1: Summary of the algorithm properties.

Our intent is to compare the performance of the four algorithms described above. The original Min_n, Mod_n and $Sketch_n$ algorithms are defined using w-shingles, i.e., the set of w contiguous terms (words), while the BSW_n algorithm uses as a unit a sliding window of w-bytes. To allow a fair comparison among all the algorithms, we use a sliding window of w-bytes in place of the w-shingle for the remainder of the paper.

○ Overall Process of Finding Similar Files

There are two essential steps in the process of finding similar files in a file collection.

The *first step* is to compute a file signature (according to any of the four similarity algorithms described above).

The *second step* is to compare file signatures for common entries and report only clusters of those files whose signature intersection is above a given similarity threshold. This step can be done in the same fashion for different file signatures and is well described in literature [4, 15, 9]. We skip a detailed description of this step due to lack of space.

There are a few performance considerations at this step. First of all, the number of entries in the files signature has an obvious impact on the processing complexity at this step. It is always useful to have a well-tuned frequency sampling parameter for a file-signature size. Second, if there are common shingles/entries that are shared by a large number of files, e.g. by more than 1000 documents, then this creates additional efficiency problems (higher processing time) during the clustering step.

3. METHODOLOGY

In order to fairly compare the four algorithms introduced in the previous section, we created a special environment for our experiments.

First, we selected 100 different HPLabs technical reports from 2007 (http://www.hpl.hp.com/techreports/2007/) and converted them to a text format. This collection, called Research Corpus (abbreviation RC_{orig}), has documents of different length as shown in Figure 1. We ordered the documents by their length: a higher *Document_ID* represents a longer document. As shown in Figure 1, our collection has documents ranging from 9 KB to 540 KB.

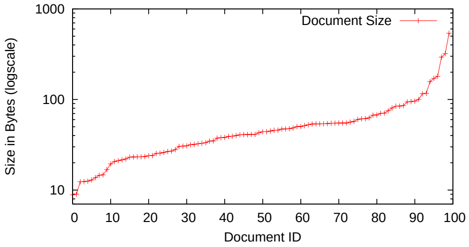

Figure 1: The document size distribution in RC_{orig}.

Second, we wrote a program that can introduce modifications to the documents from our collection in a controlled way. For example, it can either add or remove words to/from the document a predefined number of times. These modifications can be done in a random fashion or be uniformly spread through the document. The intent is to compare the four selected algorithms by using a document collection for which we have full knowledge about a number and type of modifications present in the documents.

We started with the simplest modification type: our program creates near-duplicates for each original document from RC_{orig} by inserting "a " [1] into the document 1, 2, ..., 50 times following a random distribution. Thus, the created collection RC_a^i $(1 \leq i \leq 50)$ denotes the collection where for each original document there is a near-duplicate with i-times randomly inserted "a " into the original document. For this modification type we know the "edit distance" [2] between the original and the modified document.

Let D_k and D'_k ($(1 \leq k \leq 100)$ be the original document and its near-duplicate with a controlled number of modifications i in the collection RC_a^i. We will compute the *average similarity metric* for RC_a^i as follows (remember, that there are 100 documents in the original collection RC_{orig}):

$$aver\_sim(RC_a^i) = \frac{1}{100}\sum_{k=1}^{100} sim(D_k, D'_k)$$

[1] Insertion of "a " does not have any particular meaning in our framework except that it is a clearly understood tiny modification distributed in a controlled way in the original text.

[2] The edit distance between two strings of characters is the number of operations required to transform one of them into the other.

Our goal is to compare the average similarity metric reported by the four algorithms for the created collections of near-duplicates as a function of the algorithm's parameters and the introduced modifications. In addition, we would like to understand the correlation of the reported similarity metric and the document size, i.e., whether these algorithms perform equally well on short and long documents.

4. SENSITIVITY ANALYSIS AND COMPARISON OF SIMILARITY ALGORITHMS

The four algorithms introduced in Section 2 have two explicit parameters:

- w - sliding window size (in bytes);
- n - *sampling frequency* that is used to define/impact a number of the fingerprints included in the document signature.[3]

For the algorithms Min_n and $Sketch_n$ the document signature is a fixed-size feature vector with n entries.

For the algorithms Mod_n and BSW_n the parameter n defines the frequency of fingerprint sampling, and therefore the number of entries in the document signature. The Mod_n technique selects the fingerprints whose value modulo n is zero. For BSW_n n works in a similar way: first, it identifies the fingerprints whose value modulo n is zero as a chunk boundary, and then the numerically smallest fingerprint within this chunk is selected for inclusion into the document signature.

There are performance studies that use these algorithms with widely different parameter settings. For example, in [4], 10-word shingles are used: this is approximately equivalent to an 80-100 byte sliding window. In other studies [7, 8], the authors use 5-word shingles which is approximately equivalent to a 40-50 bytes sliding window, while in [12], 6-word shingles are used.

There is a similar situation with respect to sampling frequency n that defines the number of entries that constitute a document signature. In [4], the authors use $n = 25$ for the Mod_n algorithm. In [7, 8, 12], the authors use $n = 84$ for the $Sketch_n$ algorithm, where each function f_i is a 64-bit Rabin fingerprinting function. The 84 fingerprinters use different primitive polynomials of degree 64. Therefore for this implementation of $Sketch_n$ the parameter $n \leq 84$. In our implementation of $Sketch_n$, we use Bob Jenkins' family of hash functions [14] for speed and ease of parametrization of n to any positive value.

Question: How important is the choice of sliding window size w and sampling frequency n? How sensitive is the similarity metric to different values used for these parameters?

The next two sub-sections aim to answer these and related questions.

4.1 Impact of the Sliding Window Size

The sliding window size defines the granularity for observing and sampling the contiguous terms (bytes) in the document. Intuitively, a smaller window introduces document sampling at the level of words, while a larger window produces sampling with a size closer to the sentence level. To understand the impact of the sliding window size in the four similarity algorithms under study, we use the document collections RC_a^i, $1 \leq i \leq 50$ as described in Section 3.

We compute the average similarity metric $aver\_sim(RC_a^i)$ $(1 \leq i \leq 50)$ under the four algorithms with different values for the sliding window: $window = 5, 10, 20, 40, ..., 100$ bytes.

Figure 2 shows these results for the Mod_n and BSW_n algorithms. (Since all the algorithms are very sensitive to the sliding window size and the sensitivity trends are similar, we omit graphs for Min_n and $Sketch_n$ due to lack of space).

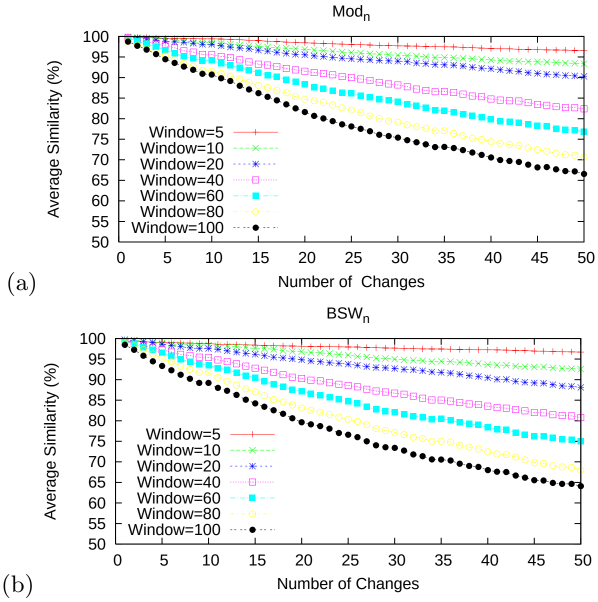

Figure 2: Impact of window size on $aver\_sim(RC_a^i)$ for algorithms: (a) Mod_{100}; and (b) BSW_{400}.

The x-axis reflects the number of changes i $(1 \leq i \leq 50)$ in the document collection under test, i.e., RC_a^i. The y-axis shows the average similarity metric $aver\_sim(RC_a^i)$ (see a formal definition in Section 3). Multiple lines in the figure show $aver\_sim(RC_a^i)$ with different values of sliding window.

First of all, the average similarity metric for a collection RC_a^i decreases as a higher number of modifications are introduced in each document. Moreover, for a larger sliding window and increasing number of modifications in the created near-duplicates, the similarity metric is much smaller than under a smaller sliding window. For example, for Mod_{100} algorithm and $window = 100$ bytes, $aver\_sim(RC_a^{50}) = 66\%$, while under $window = 5$ bytes, the average similarity metric is much higher: $aver\_sim(RC_a^{50}) = 97\%$. The trends for the BSW_{400} algorithm are the same.

Does this mean that $window = 5$ bytes is a good parameter for using in the similarity algorithms?

To make a definite conclusion, we investigate the impact of a window size on the similarity metric for the original research corpus RC_{orig}, where all the 100 documents are truly different. How much overlap or "false positive" similarity information is observed in these distinct documents while varying the sliding window size?

Figure 3 show these results for the Mod_n and BSW_n algorithms. This figure shows the CDF (Cumulative Distribution Function) of the similarity metric for all the document pairs in the RC_{orig} under different window sizes (it only shows the CDF of pairs with non-zero, positive similarity metric). We can see that algorithm Mod_n with $window = 5$ bytes identified more than 5% of the pairs having similarity above 20%, with some documents exhibiting nearly 30% similarity. Moreover, practically any pair of distinct documents does exhibit some overlap under $window = 5$, 10 bytes as shown in Figure 3. Increasing sliding window size helps to reduce the amount of overlap measured in distinct documents. In summary, if the window size is too small then there are too many "similar" fingerprints in different (dis-

[3] In this paper, we use the terms "document signature" or "feature vector" interchangeably.

tinct) documents. We can see that $window = 10$ bytes is also a bad parameter. The same observations are valid for BSW_n algorithm shown in Figure 3(b). We omit figures for algorithms Min_n and $Sketch_n$ because they show the same trend.

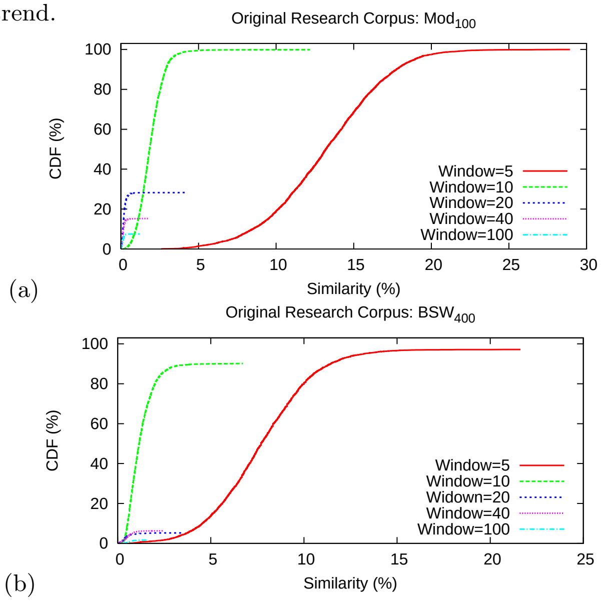

Figure 3: CDF of the similarity metric for all the document pairs in the RC_{orig}: (a) Mod_{100}; and (b) BSW_{400}.

We choose $window = 20$ bytes as a good trade-off for achieving high similarity metric for near-duplicates (Figure 2) while having a low amount of "false positive" similarity information for truly distinct documents (Figure 3).

Finally, Figure 4 shows CDF of the similarity metric for all the document pairs in the RC_{orig} under $window$=20 and the four similarity algorithms Min_n, Mod_n, $Sketch_n$, and BSW_n.

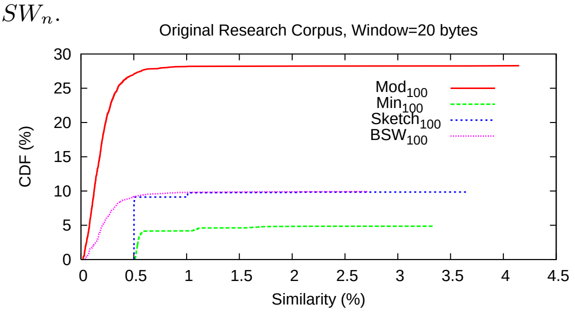

Figure 4: CDF of similarity metric in RC_{orig} with $window$=20 bytes for the four algorithms under study.

An interesting observation is that while algorithms Mod_n and BSW_n operate under a similar sampling frequency in this example, and both algorithms produce a variable-size document signature, algorithm Mod_{100} has "identified" a significantly higher number of "slightly-similar" documents: 28% of pairs are less that 4% similar under Mod_{100} while only 10% of such pairs are similar under BSW_{100}. These numbers are important for the second stage of the similarity algorithms when the clusters of similar documents are built. High percentages of "slightly-similar" documents lead to a higher computational complexity at this step.

4.2 Impact of the Sampling Frequency

The sampling frequency parameter n has a different definition in the similarity algorithms Min_n, Mod_n, $Sketch_n$, and BSW_n. For example, the algorithms Min_n and $Sketch_n$ define a fixed, n-size signature for document characterization (see Section 2 for details). The algorithms Mod_n and BSW_n produce a variable length document signature, where the number of fingerprints in the signature is proportional to the document length and dependent on n. The question is whether the choice of a sampling frequency n has a significant impact on the similarity results.

Intuitively, having only a few samples to represent the entire document might be dangerous for two (opposite) reasons. Let documents A and B have quite a few similar portions but overall be quite different. If there are only a few fingerprint samples to represent A and B, then these samples might easily "land" on the similar portions of the documents, and hence, report a high similarity for these documents while missing their difference. For the same reason, the opposite might happen: if these few fingerprint samples land on the modified portions of A and B then the same pair of documents A and B might be reported as having almost no similarity while missing that these documents do have similar portions.

Figure 5 shows $aver\_sim(RC_a^i)$, $1 \leq i \leq 50$, for $Sketch_n$ and BSW_n algorithms with different sampling frequency. $Sketch_{10}$ represents the document by using 10-feature vector, while $Sketch_{100}$ by using 100-feature vector. At first glance, the sampling frequency does not have much impact on the average similarity metric shown in Figure 5 (a).

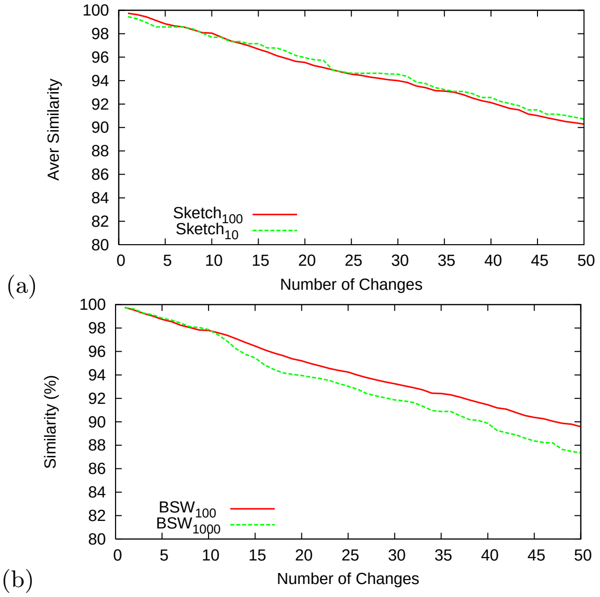

Figure 5: Impact of sampling frequency on average similarity metric: (a) $Sketch_n$, (b) BSW_n.

BSW_{1000} algorithm produces variable-size chunks (with the average chunk size around 1000 bytes), while BSW_{100} defines a finer granularity chunking with the average chunk size around 100 bytes. Given a 10 KB document, BSW_{1000} will produce a document signature with approximately 10 entries, while BSW_{100} will have ∼100 entries. Again, at first glance, this difference does not seem to have much impact on the average similarity metric shown in Figure 5 (b).

Now, let us have a closer look at the similarity of the original and near-duplicate documents in the collection RC_a^{50} under different sampling frequency parameters. Remember, the original document and its near-duplicate in RC_a^{50} are differentiated by the random insertion of "a " in 50 places in the document. Figure 6 shows these results. The similarity metric reported by $Sketch_{100}$ is much more consistent

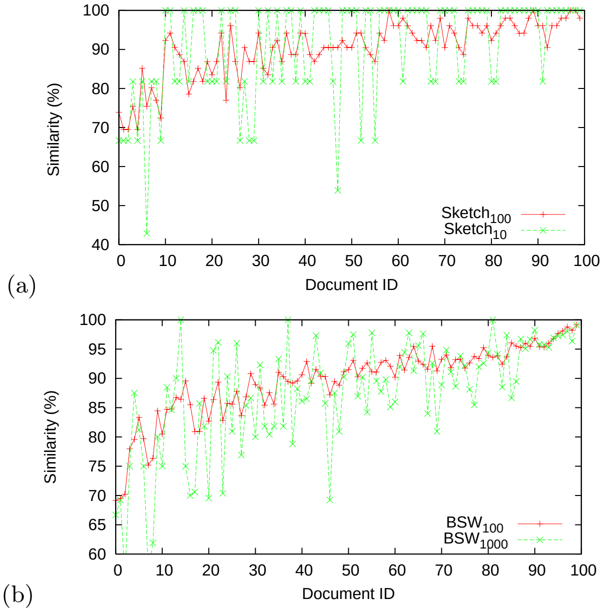

Figure 6: Similarity of the original and near-duplicate documents in RC_a^{50} under different sampling frequency: (a) $Sketch_{10}$ vs $Sketch_{100}$, (b) BSW_{1000} vs BSW_{100}.

than the similarity metric reported by $Sketch_{10}$ for the same pairs of documents. In many cases, $Sketch_{10}$ misses all differences in the pairs of near-duplicates and reports 100% similarity values as shown in Figure 6 (a). At the same time, we can see that there are situations when $Sketch_{10}$ reports much lower similarity metric compared to $Sketch_{100}$, because these few sampled fingerprints "land" on the modified portions of the near-duplicate documents. Figure 6 (b) shows the same trend for BSW_{1000} and BSW_{100}. Using a coarser chunking leads to the same high variance in the similarity results.

Note that for shorter documents (remember, that documents with smaller ID represent shorter documents) the impact of the sampling frequency in the presence of significant number of "small" modifications is especially pronounced. Due to lack of space we omit graphs for Min_n and Mod_n which exhibit similar trends.

4.3 Comparison of the Similarity Algorithms

In this section, we compare the performance and accuracy of the four similarity algorithms. Using guidance from Sections 4.1 and 4.2 on the sensitivity of the algorithms to sliding window size and frequency sampling, we choose *window* = 20 bytes, and n=100 for Min_n and $Sketch_n$.

Note, that Min_{100} and $Sketch_{100}$ produce a fixed 100-feature vector for any document in the set, while Mod_n and BSW_n generate a feature vector that is proportional to the document size: the rule of thumb is that if document D is S_D bytes long then its feature vector has on average S_D/n entries. Since the average file size in the research corpus is ∼50 KB, we choose n=500 for Mod_n and BSW_n. This way the average feature vectors of different algorithms under study are approximately the same (for the Research Corpus used in our experiments).

Figure 7(a) shows the average similarity metric for RC_a^i $(1 \leq i \leq 50)$ under the four algorithms Mod_{500}, Min_{100}, $Sketch_{100}$, and BSW_{500}. For a small number of changes in the near-duplicate documents (less than 10) the average similarity metric reported by all four algorithms is nearly the same. Then with an increasing number of changes in the near-duplicate documents the reported average similarity metrics under different algorithms start to diverge as shown in Figure 7(a). In order to verify the validity of these trends, we performed additional experiments where the near-duplicates are generated according to different modifications of the original documents.

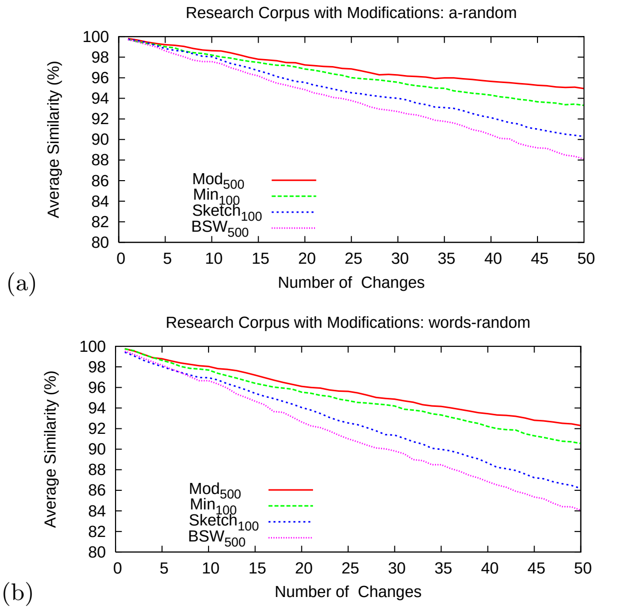

Figure 7: Average similarity metric under the four similarity algorithms: (a) using RC_a^i and (b) using RC_{words}^i.

Figure 7(b) shows the average similarity metric computed for a case when near-duplicates are generated by a random removal of a word from the original document (1 to 50 times accordingly). While the values of average similarity metric are slightly lower (due to a higher "edit"-distance between the original and the modified documents), the trends of Figures 7(a) and (b) are the same. (In fact, we performed other experiments with a different distribution of changes. Uniformly distributed changes have a stronger negative impact on the similarity metric, while we still observe similar trends between the algorithms).

The summary of Figure 7 is that for the same set of near-duplicate documents Mod_n and Min_n report on average a higher (more accurate) similarity metric than algorithms $Sketch_n$ and BSW_n.

To analyze the divergence of similarity metrics under different algorithms in more detail, we plotted a similarity metric for pairs of original and modified documents under different number of modifications as shown in Figure 8. While unfortunately these graphs are "very busy" they convey well the overall trend. When the modified document has a relatively small number of changes (less than 10), the performance of all the algorithms is alike across different document sizes as shown in Figures 8 (a). However, under a higher number of changes in modified documents, the algorithms $Sketch_n$ and BSW_n "reflect" more of such changes in their feature vectors, especially for shorter documents as shown in Figures 8 (b).

An interesting observation is that $Sketch_n$ and BSW_n are the representatives of somewhat different approaches: $Sketch_n$ produces a fixed-size feature vector while BSW_n generates a feature vector that is proportional to the document size. However, they both are more sensitive to the document modifications than Mod_n and Min_n.

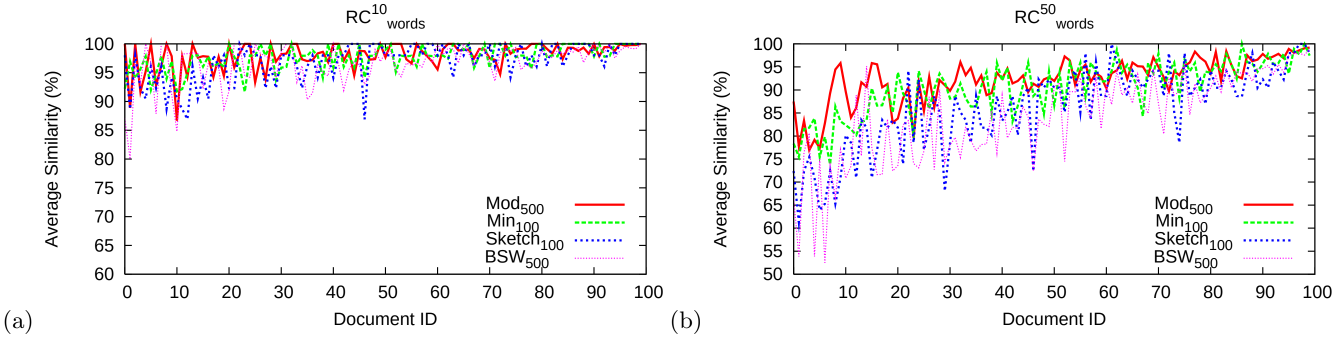

Figure 8: Similarity of the original and modified documents under the four similarity algorithms: (a) using RC^{10}_{words} and (b) using RC^{50}_{words}.

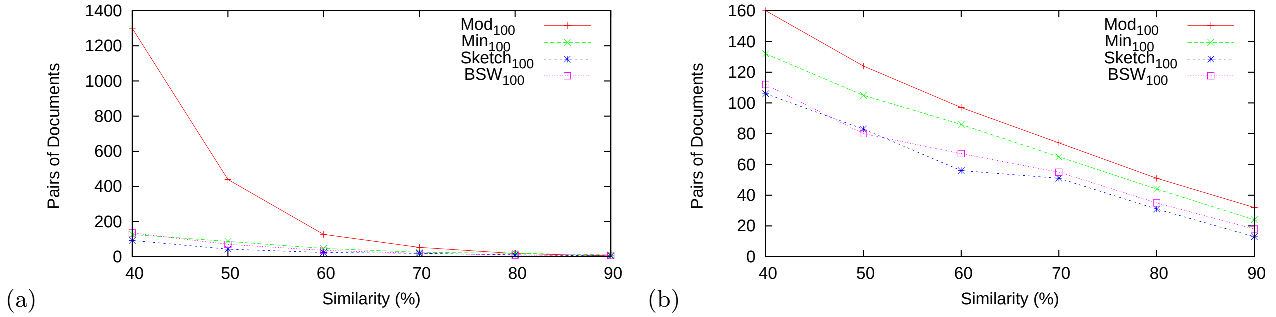

Figure 9: Number of similar file pairs identified under different similarity algorithms in the enterprise repositories: (a) Collection_1 and (b) Collection_2.

5. FINDING SIMILAR DOCUMENTS IN ENTERPRISE REPOSITORIES

Hewlett-Packard has large collections of technical support documents on different topics. As a part of content management it is important to identify and filter out support documents that are largely duplicates of newer versions in order to improve the quality of the existing collection.

Our goal was to process given collections of documents with the four similarity algorithms under study and compare the results.

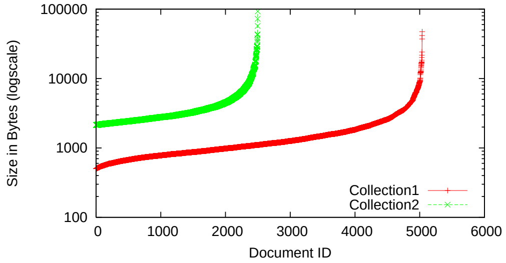

Figure 10: The document size distribution in two enterprise repositories.

We processed two collections of documents with 5040 and 2500 documents respectively. The document length distributions for both collections are shown in Figure 10. The documents in the first collection were relatively short (600-1500 bytes) with a small subset of longer documents. The second collection had slightly longer documents (2-4 KB) and a small subset of long (larger than 10 KB) documents.

Since these two collections had mostly short files, we tuned a sampling frequency in Mod_n and BSW_n to produce finer granularity by using n=100. Even then, Mod_{100} and BSW_{100} had 7 times fewer (on average) hashes in the document signature than $Sketch_{100}$ or Min_{100} ($Sketch_{100}$ and Min_{100} produce 100-entry feature vectors independent of the file size). For the second collection this ratio was 2.5 times.

Figure 9 shows the number of similar file pairs for varying the similarity theshold from 40% to 90% identified under different similarity algorithms in the two enterprise collections. Typically, the pairs of files with similarity metrics higher than 70% might be good candidates for being near-duplicates. We analyzed pairs of documents with similarity above 70% reported by different algorithms, and Table 2 presents the summary of the results. For both collections,

	Collection_1			
	Mod_{100}	Min_{100}	$Sketch_{100}$	BSW_{100}
Total (65)	61	25	17	20
Correct (51)	48	24	17	19
	Collection_2			
Total (81)	75	65	49	55
Correct (81)	75	65	49	55

Table 2: Summary of the results for *Collection_1* and *Collection_2*.

Mod_{100} and Min_{100} identified a higher number of potentially similar pairs compared to $Sketch_n$ and BSW_n. At the same time, for *Collection_1*, the combined number of potentially similar pairs reported by the four algorithms independently is 65. However, only 12 of them are reported by all 4 algorithms. For *Collection_2*, the combined number of pairs reported by all four algorithms independently is 81, but only 45 of them are reported by all 4 algorithms.

The trends in reported results agree with our observations for the four algorithms derived with the Research Corpus experiments (except a closer match between BSW_n and $Sketch_n$). However, in the case of the two enterprise collections we do not know the "ground truth", i.e., which of the reported documents are indeed similar, and which ones might be false positives. To understand this we performed an additional document analysis in the following way. For each of the reported similar pair of documents D and D', we used UNIX *diff* utility to compute the edit distance. In most of the cases, the outcome of *diff* was very compact with easily observable difference in the pair of documents due to document number, or date, or some other small details. In these cases, it was clear that documents D and D' were indeed similar. In a few cases, the outcome of *diff* was significant and, indeed, documents D and D' were apparently different. Using this approach, we identified which of the similar pairs were reported correctly and which were false positives.

It was interesting to see the groups of similar documents and what constitute overlapping and different portions of these documents. For example, there were pairs of technical support documents with different dates and document numbers, but very similar root cause analysis and solution information written about different applications, say X and Y. After talking to experts it became apparent that Y was a new generation of product that replaced X.

As shown in Table 2, for the first collection, Mod_{100} had highest number of correctly identified documents (48), but at the same time 13 documents ($13 = 61-48$) were false positives. Min_{100} and BSW_{100} both had 1 false positive pair. While $Sketch_{100}$ did not have any false positives, it had the lowest number of reported similar pairs among the four algorithms. All the false positives happened for short documents. One of the explanations is that the documents in both collections had a "loose" configuration template defining content of the documents, and for very short documents it introduced by itself a significant amount of "noise". For the second collection of documents, where the documents were longer, none of the algorithms reported false positives as shown in Table 2.

In summary, Mod_n significantly outperformed the other algorithms, with Min_n being second, BSW_n and $Sketch_n$ sharing the third place. These findings are consistent with our observations and experiments in Section 4 with the Research Corpus.

An interesting extension of this work will be incorporating some additional document analysis for identified pairs of similar documents to avoid false positives and helping experts to more easily filter them out. Once the clusters of similar documents are built, one can apply more expensive methods to "highlight" the exact difference in identified pairs of documents to semi-automate the document analysis process at this stage.

6. RUNTIME COMPARISON

We performed a comparison of the document processing time under different algorithms and their parameters. Figure 11 shows the processing time of a 500 KB file under Mod_n, Min_n, $Sketch_n$ and BSW_n. While this figure is specific to our implementation of these algorithms, it still reflects some general trends. Document processing under Mod_n, Min_n, and BSW_n is based on Rabin fingerprints [17] that have a very fast software implementation and are independent of a sliding window size. Processing time under $Sketch_n$ algorithm involves concurrent computation with n different hash functions (we used the Bob Jenkins hash function for speed). Moreover hash computation over a larger sliding window has a significant additional overhead.

The results in Figure 11 are for a longer file to stress the difference. Obviously, for short files this difference is less pronounced. When choosing a particular similarity algorithm, one needs to carefully estimate the length of the file signature as well as the document processing cost based on the average file size. However, in general, the processing cost under Mod_n, Min_n, and BSW_n is lower than under $Sketch_n$. The $Sketch_n$ algorithm must compute a family of n hash functions rather than just a single hash function, and this significantly impacts its speed.

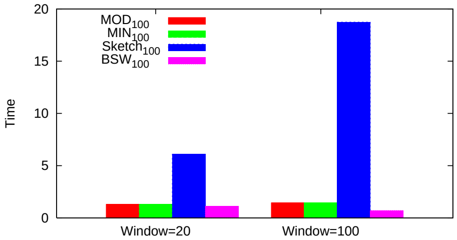

Figure 11: Comparison of the processing time for a 500 KB file.

7. RELATED WORK

Many document repositories, such as newswire, web sites, and digital libraries typically have a significant amount of repeated or similar information. The extent to which documents are considered similar to each other defines a similarity spectrum. On one side there is a broad topical (semantic) similarity with document identity on the other side. In this paper, we studied a few algorithms that can be classified as a syntactic similarity group. The similarity of documents under this definition is entirely defined by the syntactic properties of the documents.

Duplicate and near-duplicate web pages cause problems for web search engines: increased index size , slower searches, and lower quality results. A variety of syntactic similarity algorithms proposed for detection of near-duplicate web pages are based on a "shingling" technique [2, 3]. While these algorithms were proposed more than 10 years ago, we are not aware of any performance study comparing them, especially in the context of enterprise information management. In [12], M. Henzinger compares the performance of the $Sketch_n$ algorithm with Charikar's algorithm [5] based on random projections of words in a document. More exactly, a variation of the $Sketch_n$ algorithm with super-shingles is used in this work. Under this approach a set of shingles is combined into a super-shingle to simplify the second step – clustering process. It is interesting work that involves a large-scale evaluation, namely a set of 1.6B distinct web pages. The author finds that neither of the algorithms works well for finding near-duplicate pairs on the same site, while both achieve high precision for near-duplicate pairs on different sites. In this work, the author chooses a set of parameters for these algorithms based on the earlier use cases. It might be interesting to repeat the same algorithms under different parameters and compare the outcome. In [7], the authors apply $Sketch_n$ algorithm with super-shingles for identifying near-duplicate web pages but pursue a different goal of understanding how the clusters of near-duplicates evolve over time. Another interesting performance study, [19], analyzes the sources and amount of document replication on the web.

There is a group of similarity algorithms that aim to detect copyright violations and plagiarism [1, 18, 10, 13]. In [18], the authors study a variety of choices for chunking policies (fixed size, variable size, small vs. large chunks, overlapping vs. non-overlapping chunks) where the chunks are defined at the granularity of words, group of words, and sentences. The authors compare the cost (in terms of storage) of different chunking schemes and accuracy of finding the overlapping chunks in documents from DL. The chunking approach in their paper is closer to a classic Basic Sliding Window algorithm with a difference that a "word" is the basic unit (compared to a byte in our approach). In [9], authors successfully applied the content-based chunking algorithm to identify similar documents in large enterprise repositories. However, in [18, 9], the authors use the entire chunk hashes for a file signature (compared to the smallest hash of the window within the chunk in our approach). This is *exact* document fingerprinting; all the policies considered in our paper use *approximate* fingerprinting. They find that a coarser chunking misses many overlaps within the test set, while a finer granularity chunking improves the outcome. In [13], the authors concentrate on identifying plagiarism and discuss how this problem is different from a more traditional problem of finding similar documents. They investigate a number of approaches that can be applied to this problem, and mostly concentrate on the *1-to-n* problem, i.e., finding the derivatives for a given document. A good survey of different methods proposed for finding similar documents to a given one is provided in [6]. For solving the *n-to-n* similarity problem the authors promote an *I-match* approach [6] which is based on term collection statistics. It is closer to a semantic document analysis rather than a pure syntactic approach. *I-match* identifies a smaller number of near-duplicates (but with a higher accuracy) compared to a shingling technique because it requires an exact match for the document terms remaining after the syntactic filtration process.

8. CONCLUSION

Many enterprises are building new applications associated with document management, data retention, regulatory compliance, and litigation issues as a part of information management solution. The problem of robust discovery of similar enterprise documents is an essential part of this solution. In this paper, we evaluate and compare the performance of four syntactic similarity algorithms. Three algorithms are based on Broder's "shingling" technique while the fourth algorithm employs a more recent approach, "content-based chunking". Our performance study reveals that the similarity metric reported by all four algorithms is highly sensitive to settings of the sliding window size and sampling frequency parameters. We identify a useful range of these parameters for achieving good practical results, and compare the performance of the four algorithms in a controlled environment. We validate our results by applying these algorithms to finding similar documents in two large collections of HP technical support documents.

In this work, we used a slightly modified version of a traditional Basic Sliding Window algorithm: instead of taking the hash of the entire chunk, we used the numerically smallest fingerprinted sliding window within this chunk. This modification made the original algorithm more resilient to small changes within the chunk, and improved its performance for small documents. Additionally, this change enabled a fair comparison between the four similarity algorithms since they were defined using the same type parameters such as the sliding window size and sampling frequency.

For a family of *shingling*-based algorithms the sampling is essential, because it drastically reduces the amount of data (fingerprinted shingles) to compare and makes the whole approach feasible. However, sampling always introduces a degree of "fuzziness" and a probability for false positives.

The traditional Basic Sliding Window algorithm produces a compact file signature without sampling by using the chunk hashes. This file signature encodes exact information about the file. If two files do have 9 out of 10 hashes in common, then we know precisely which portions of the files are identical. This is an appealing property. We believe that traditional Basic Sliding Window algorithm might be a more promising approach for longer files than the "sampling"-based technique and is worth future investigation.

9. REFERENCES

[1] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanism for digital documents. Proc. of ACM SIGMOD International Conference on Management Data, May, 1995.

[2] A. Broder. On the Resemblance and Containment of Documents. Proc. of IEEE Conf. on the Compression and Complexity of Sequences, June, 1997.

[3] A. Broder, M. Charikar, A. Frieze, M. Mitzenmacher. Min-wise independent permutations. Proc. of the Thirtieth Annual ACM Symposium on Theory of Computing,q Dallas, Texas, USA, May, 1998.

[4] A. Broder, S. Glassman, M. Manasse, G. Zweig. Syntactic clustering of the Web. Selected papers from the 6-th Intl. Conference on World Wide Web, Sept.1997, Santa Clara, CA, USA.

[5] M. S. Charikar. Estimation technique from rounding algorithms. Proc. of 34th Annual ACM Symposium on Theory of Computing (STOC), May, 2002.

[6] A. Chowdhury, O. Frieder, D. Grossman, and M. McCabe. Collection Statistics for Fast Duplicate Document Detection. ACM Transactions on Information Systems (TOIS), 20(2), April 2002.

[7] D. Fetterly, M. Manasse, M. Najork. On the Evolution of Clusters of Near-Duplicate Web Pages. Journal of Web Engineering, vol.2, N.4, Oct, 2004.

[8] D. Fetterly, M. Manasse, M. Najork. Detecting Phrase-Level Duplication on the World Wide Web. Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05), Salvador, Brazil, Aug, 2005.

[9] G. Forman, K. Eshghi, S. Chiocchetti. Finding Similar Files in Large Document Repositories. Proc of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2005.

[10] N. Heintze. Scalable document fingerprinting. Proc. of the Second USENIX Workshop on Electronic Commerce, 1996.

[11] V. Henson. An analysis of compare-by-hash. Proc. of the 9th conference on Hot Topics in Operating Systems (HotOS'03), Lihue, Hawaii, USA, 2003.

[12] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. Proc. of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06), Seattle, Washington, USA, 2006.

[13] T. Hoad and J. Zobel. Methods for Identifying Versioned and Plagiarised Documents. Journal of the American Society for Information Science and Technology, vol. 54, 2002.

[14] B. Jenkins. Hash Functions. "Algorithm Alley", Dr. Dobb's Journal, September, 1997, http://www.ddj.com/184410284

[15] U. Manber. Finding similar files in a large file system. Proc. of the Winter 1994 USENIX Conference, 1994.

[16] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP'01), Banff, Canada, October 2001.

[17] M.O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[18] N. Shivakumar and H. Garcia-Molina. Building a Scalable and Accurate Copy Detection Mechanism. Proc. of 1st ACM International Conference on Digital Libraries , March 1996.

[19] N. Shivakumar and H.Garcia-Molina. Finding near-replicas of documents on the web. Proc. of Workshop on Web Databases (WebDB'96), Valencia, Spain, 1996.

[20] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Australian National University, 1996.