Copy Detection Systems for Digital Documents

Douglas M. Campbell, Wendy R. Chen, Randy D. Smith
Brigham Young University
Department of Computer Science
Provo, UT 84602
{campbell,wrchen}@cs.byu.edu, smithr@azwest.net

Abstract

Partial or total duplication of document content is common to large digital libraries. In this paper, we present a copy detection system to automate the detection of duplication in digital documents. The system we present is sentence-based and makes three contributions: it proposes an intuitive definition of similarity between documents; it produces the distribution of overlap that exists between overlapping documents; it is resistant to inaccuracy due to large variations in document size. We report the results of several experiments that illustrate the behavior and functionality of the system.

1. Introduction

In large digital libraries, of which the Internet is a prime example, partial and/or total duplication of document content is common. Duplication of content arises for a variety of reasons. To name a few: data may be mirrored at multiple sites; documents may be unknowingly submitted multiple times; a single document in the library may be the concatenation of several smaller documents in the library; documents may be revisions of each other; documents may be plagiarized versions of other documents.

Successful detection of duplicate content is important to the long-term success of digital libraries, the Internet, and digitally distributed media in general. Three issues illustrate this:

- From the perspective of information retrieval, duplication degrades the efficiency of the retrieval process.
- From the perspective of electronic commerce, duplicated copyrighted material is a constant source of lost revenues for copyright holders (consider the large amount of illegally distributed music encoded in "mp3" format available over the Internet as a case in point).
- From the perspective of dynamic digital libraries such as the Internet, duplicates provide alternative copies of a document when an original is removed.

Copy detection systems, which automate the process of detecting overlap between electronic documents, are a recent addition to digital library technology. Currently, there are three major copy detection systems: COPS, SCAM, and KOALA. These systems share the following weaknesses:

- Each system struggles to measure the amount of overlap existing between two documents.
- Each system fails to provide any information describing where overlap occurs between documents.
- Two of the three systems, COPS and SCAM, are plagued with false positives (which occur when two documents are falsely reported to have overlap).
- Two of the three systems, SCAM and KOALA, have trouble detecting overlap between documents that vary widely in size.

In this paper, we present a copy detection system that addresses each of the preceding four weaknesses. Our copy detection system performs copy detection using a fingerprinting technique centered around the sentence structure of documents. Given a document D to test, our system hashes each sentence in D. It finds overlap by comparing the hash values in D with the hash values contained in a database of document hash values. Relative positions of sentences in both the test document D and the database documents are then used to determine the locations where overlap occurs between documents.

Our system incorporates advantageous properties of existing systems, leaves behind many of their weaknesses, and contains capabilities not available in existing systems. Specifically, our copy detection system exhibits the following characteristics:

- It incorporates an intuitive definition of similarity between documents. Documents with no sentences in common are 0% similar, and identical documents are 100% similar.
- It is able to compute both the amount of overlap that

exists between documents as well as display the location of overlap between two or more documents.
- It is robust over large variations in the sizes of documents.

In addition, as part of the fingerprinting (hashing) technique, we present a scalable, near-perfect hash function for English sentences that can hash millions of sentences while giving up only tens of collisions.

2. Applications and relations to conference

We present seven Internet-based applications that need a copy detection system.

1. Internet web crawling. Broder, et. al. [4] gathered 30,000,000 web documents during 1997 and discovered that 20% were identical copies of existing documents [10][18]. Duplicate Linux documentation alone had a combined file size of 4.5 gigabytes [18]. A web crawler that can determine a duplicate page can save processing time and reduce search engine index size.
2. Internet e-commerce. E-commerce services must guard against losses caused by illegal distribution of a copied electronic product. Garcia-Molina, Ketchpel, and Shivakumar [6] give four steps to control such illegal duplication, the central component of which is a copy detection system.
3. Internet fault tolerance. The average World Wide Web hyperlink lasts only 44 days [9]. "404 Document Not Found" is a plague on retrieval efforts. Instead of not processing duplicate pages (as in (2) above), when a crawler determines that a page is a duplicate, the search engine can be instructed to store the URL of the duplicate to increase fault tolerance.
4. Internet evolution. How do web pages evolve? How frequently does a page change? How much does a page change within a time step? How often does a page move between servers and within a server? What is the typical life cycle of a page? What are the access patterns of a page? To answer these questions, a crawler can use a copy detection system to periodically reprocess pages to measure changes. Understanding how Internet pages evolve leads to better servers, crawlers, browsers, search engines, and web tools.
5. Internet paper submissions. Editors face two onerous tasks concerning submissions. (1) Determining overlap between a submission and previous publications. (2) Reviewing all changes between a revision and the original. A copy detection system automates these two tasks. It automatically reports the submission's similarity to previous work. It automatically lists all sentences changed between a revision and the original. Naturally the editor's judgment is still required, but a copy detection system can reduce the tedium.
6. Internet information retrieval. Search engines find documents that match a query. Identical documents have identical rankings. Therefore, search engines will return a list of different URL's, which, when retrieved, turn out to be for the same document. A copy detection system can improve information retrieval and reduce Internet congestion by eliminating a search engine's unintentional listing of duplicate documents.
7. Internet copyright enforcement. Let D be a document whose copyrighted material is to be enforced. A violator may have copied D in its entirety or tried to hide the violation by taking only occasional sentences in a non-ordered fashion. A sentence-oriented copy detection system identifies each copied sentence, even when in they are in a (scattered) random order.

3. Related work

This section discusses six systems and techniques used for detecting overlap between documents. These systems are *diff*, SIF, COPS, SCAM, KOALA, and Digital Watermarking.

3.1. *diff*

diff applies the longest common subsequence algorithm to find differences between two text documents. It is commonly used for source code, lists, and other line-oriented files. In the context of document copy detection, though, *diff* has two weaknesses. First, it is subject to phase shifts. In a textual document, which is not explicitly line-oriented, the addition of a few words at the beginning of a document can cause every line to shift a few words to the right (in order to stay within formatting margins). Although the modified document is virtually identical to the original document, *diff* would find nothing in common between the two documents since the shift causes each line in the modified document to be altered. Second, a longest common subsequence algorithm can only find lines that are in the same order. For an extreme example, let *D* be a (line-oriented) document, all of whose lines are unique. Let *C* be the same document with all the lines of *D* in reverse order, e.g., from bottom to top. Then although every line of *C* is also in *D*, *diff* reports that they have *at most one line in common*.

3.2. SIF

SIF [10] was developed in 1994 as a tool useful for finding similar files in a file system. It uses a fingerprinting scheme developed by Rabin [14] to characterize documents. A fingerprint of a string *S* is an integer f(*S*) with the property that if *S'* is even a slight modification of *S*, then with very high probability f(*S*) and f(*S'*) differ. SIF generates fingerprints from randomly selected substrings of a document. Documents are deemed similar if the number of common fingerprints exceeds a specified threshold. SIF was originally developed as a tool for file management. In the scope of copy detection, it has several weaknesses. First, its notion of similarity is not well suited to copy detection. Second, it can't gauge the extent of document overlap. Finally, it does not display the locations of similar documents in which overlap occurs.

3.3. COPS

COPS [3], developed in 1995, was designed specifically to detect plagiarism between documents. It incorporates a hash-based registration scheme to perform copy detection. Documents are registered by hashing each sentence in the document and storing the hash values in a database. Plagiarism detection is performed by comparing the hash values of a test document to the hash values in the database. Like SIF, COPS reports a "violation" if the number of common hash values exceeds a given threshold. From an evolutionary standpoint, COPS adds to SIF a better technique for estimating similarity. Whereas SIF compares only selected subsets of documents, COPS has the capacity to compare each sentence (via its hash value) of a document with all other registered documents. COPS has three weaknesses. First, the hash function produces a large number of collisions, which falsely raise the amount of similarity between compared documents. Second, COPS can only detect plagiarism for documents that have at least 2% (10 sentences or more for their test sets) overlap. Third, COPS' sentence disambiguation often selects incorrect sentence boundaries, which the authors report as the source of some incorrect matches. Our system overcomes each of these three weaknesses.

3.4. SCAM

SCAM [15][16][17], developed in 1996, uses information retrieval techniques to perform word-based copy detection. For small documents, SCAM identified more overlap than COPS, but it also had more false positives. SCAM has the following four characteristics: First, SCAM is geared towards small documents that average 5KB in size [19]. Additionally, no performance analysis is given for larger documents. Second, the similarity measure used (which is a modified cosine similarity measure) is ill-defined; it can report non-identical documents as being 100% similar. Third, SCAM cannot provide the location of overlap between documents. Finally, its storage requirements are high. Storage requirements for each processed document range from 30% to 65% of the size of the original document.

3.5. KOALA

KOALA [8], developed in 1996, specifically targeted plagiarism. It struck a compromise between SIF's random fingerprinting and COPS' exhaustive fingerprinting by selecting substrings of a document based on their usage frequency. The result of this is a reduction in storage requirements without sacrificing the accuracy of the system. Although KOALA provides high levels of accuracy even in the presence of small amounts of overlap between documents, it does have the following two weaknesses: First, it provides only document-level similarity; it can not report the location of overlap between two documents. Second, it is not robust over documents of widely varying sizes.

3.6. Digital watermarking

Digital watermarking refers to the technique of lacing digital documents with additional information. Digital watermarks may be easily applied to audio, video, and still image data, and may be either visibly noticeable or imperceptible to the naked eye. In the realm of textual document copy detection, digital watermarking uses techniques such as line-shift coding, word-shift coding, and feature coding to imperceptibly place additional information in a document. Although a considerable amount of work has been done on how to protect watermarks from attack [1][2][12], digital watermarking of text documents has two weaknesses. First, it offers *no protection* when the formatting of the document is removed. Second, it cannot reliably detect partial overlap between documents.

4. Test collections

At the core of our copy detection system is a sentence-based hash function. In the process of developing a hash function, we identified the following three requirements: first, the hash function must have near-perfect behavior (see Section 5). Second, it must be scalable. Third, it must be resistant to changes in document *genre*. Consequently, we needed sets of documents that allowed us to test these criteria.

Our first attempt was to build a 400-megabyte document collection by hand using the Internet for source material. We failed for two reasons: first, web documents are not necessarily sentence-based. They often contain formatting, html, JavaScript functions, lists, tables, and hyperlinks. Second, we observed that the textual content of web documents averages 5 kilobytes in length. Obtaining a 400-megabyte collection of data would require us to manually examine 80,000 documents for suitability.

For our tests, we created four document collections that average approximately 400 megabytes per collection. The first three collections, *wsj*, *ap*, and *zd*, were created from data used for the TREC conferences [22]. The fourth collection, *gutenberg*, was created from resources compiled from *The Gutenberg Project* [21]. Descriptions of the collections are as follows:

- *wsj* (Wall Street Journal articles, 1987-1992), 2,911,293 sentences, 435 megabytes.
- *ap* (Associated Press newswire articles, 1988-1989), 3,575,343 sentences, 452 megabytes.
- *zd* (Ziff-Davis technically oriented computer material, 1989-1990), 2,356,795 sentences, 290 megabytes.
- *gutenberg* (from Project Gutenberg [20] 1996-1998), 4,113,089 sentences, 412 megabytes.

We used Campbell's syntactic sentence algorithm [5] to disambiguate the documents into their component sentences [13].

5. A Near-perfect hash function for English sentences

Our copy detection is centered around a sentence-based hash function. This hash function bears the burden of accuracy and performance. While many existing hashing algorithms tolerate collisions to obtain a uniform distribution of hash values, hash collisions in a copy detection system are a serious problem. In a copy detection system, hash collisions are false positives; they assert that sentences are identical when in fact they are distinct.

We define a hash function to be near-perfect if and only if it averages no more than one collision for every 100,000 sentences. In addition, a copy detection system demands that the hash function be scalable and resistant to changes in document *genre*. We consider a hash function to be scalable if the ratio of the number of collisions to the number of sentences tends to a constant as the number of sentences grows. For our purposes, we require scalability to millions of sentences.

We consider a hash function to be resistant to changes in document *genre* if the hash function suffers little degradation over different document *genres* (such as technical papers, newspaper articles, magazine articles, and literary works.)

We comment that perfect hash functions are not applicable to our system. Although perfect hash functions are often used in building information retrieval systems, they are best suited for a static key (sentence) set. With a dynamic key set, each addition of a key (sentence) to the set of keys requires that the hash values for the entire set be re-computed. This requires at best O(*mn*) time (*m* is the number of sentences, *n* is the number of additions over time) with the constant coefficients quite large. Furthermore, a perfect hash function gives us better results than we need; it suffices for us to make the occurrence of collisions rare provided we can handle them efficiently when they are encountered.

Evolution of our near-perfect hash function went through five stages. In each of the following functions, let *S* be a sentence.

5.1. Hash 1

Our first hash function was based on Heintze's algorithm [8]. It converts *S* to lower case; removes all non-consonants from *S* and returns the first four remaining consonants. When applied to 400,000 sentences, 20,000 buckets contained collisions.

5.2. Hash 2

Our second hash function was based on classical multiplication and modulo arithmetic hash functions [11]. It converts *S* to lower case, removes all non-alphabetic characters from *S*, divides *S* into 4-character substrings, and returns a hash value as follows:

```
hash = 0;
FOR i = 1 TO length (S) in steps of 4 DO
    hash = (hash + i • ($S_i \cdot 26^3 + S_{i+1} \cdot 26^2 +$
                $S_{i+2} \cdot 26^1 + S_{i+3}$));
RETURN (hash MOD 1299827);
```

The number 1299827 is a prime number larger than the number of sentences applied to the test. When applied to 400,000 sentences, 31,000 buckets contained collisions.

5.3. Hash 3

Our third hash function was based on the 4-grams of a sentence. The set of 4-grams of a sentence *S* is the set of all substrings of length 4 in *S*. For example, the 4-grams of the string "great expectations" is the set {grea, reat, eate, atex, texp, expe, xpec, pect, ecta, ctat, tati, atio, tion, ions} (space ignored).

We gathered, tallied, and sorted over 615,000,000 4-grams. The hash function for a sentence simply returned the three least frequent 4-grams in the sentence

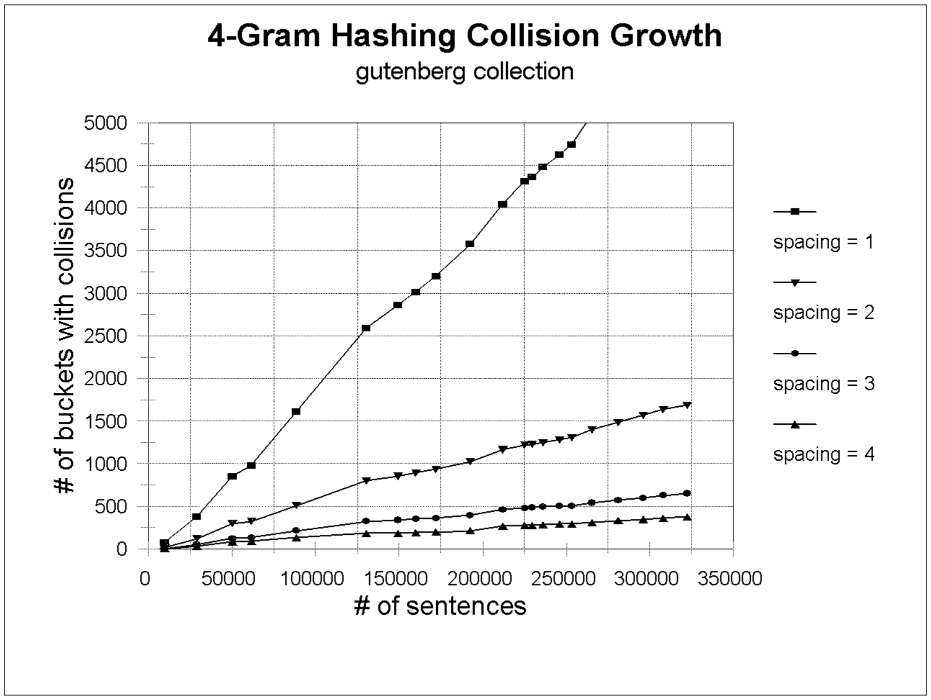

Figure 1 Reduction in collisions due to the successive increase in spacing constraints.

concatenated in order of appearance, as determined by the statistics we generated for the 615,000,000 4-grams. When applied to 400,000 sentences, only 6000 buckets had collisions.

Hash 3 had another important trait: it provided us with some ability to control the behavior of the distribution. For example, the addition of a fourth 4-gram to the hash function would produce a distribution that is strictly better (i.e., closer to uniform) distribution (or, at least it will not create a distribution that is any worse than just using three 4-grams). This is the property that is exploited to further drive development of the hash function that we use.

5.4. Hash 4

Our fourth hash function was based on the three least frequent 4-grams in a sentence concatenated in their order of appearance that also satisfied imposed *spacing constraints*. In particular, we required that the least frequent 4-grams comprising the hash function be separated by at least *d* characters in the sentence. We gathered voluminous experimental data as *d* ranged from 1 to 4. The value 3 was found to produce optimal results: when applied to 400,000 sentences, 500 buckets had collisions. Figure 1 shows the reduction in collisions for a portion of the *gutenberg* collection that occurs as the spacing constraints are increased (note that although a spacing of *d*=4 produces fewer collisions, it is the beginning of diminishing returns in which higher values of *d* impact the hash function in other undesirable ways).

5.5. Hash 5

Our final hash function was based on concatenating the three least frequent 4-grams (with a spacing of *d*=3) with a second two-byte hash value that represents other sentence characteristics. When applied to 400,000 sentences, only 2 buckets had collisions. We tested Hash 5 on our four 400-megabyte *gutenberg*, *wsj*, *zd*, and *ap* collections. Figure 2, scaled to 2.3 million sentences, shows that Hash 5 hashed millions of sentences while suffering only tens of collisions. Table 1 shows the results for the full-sized collections. We observed that Hash 5 performed less well on the *ap* collection, which is composed of news stories. Since news stories frequently access the same sources and borrow liberally from each other, we conjectured that the *ap* collection contained many more "nearly identical" sentences than the other three collections. This conjecture was confirmed by Experiment 1 in Section 9.

In conclusion, we found Hash 5 to be suitable for our copy detection system. In particular, Hash 5 satisfies the three requirements of near-perfect behavior: only tens of collisions for millions of sentences, scalability to millions of sentences, and robustness over multiple document *genres*.

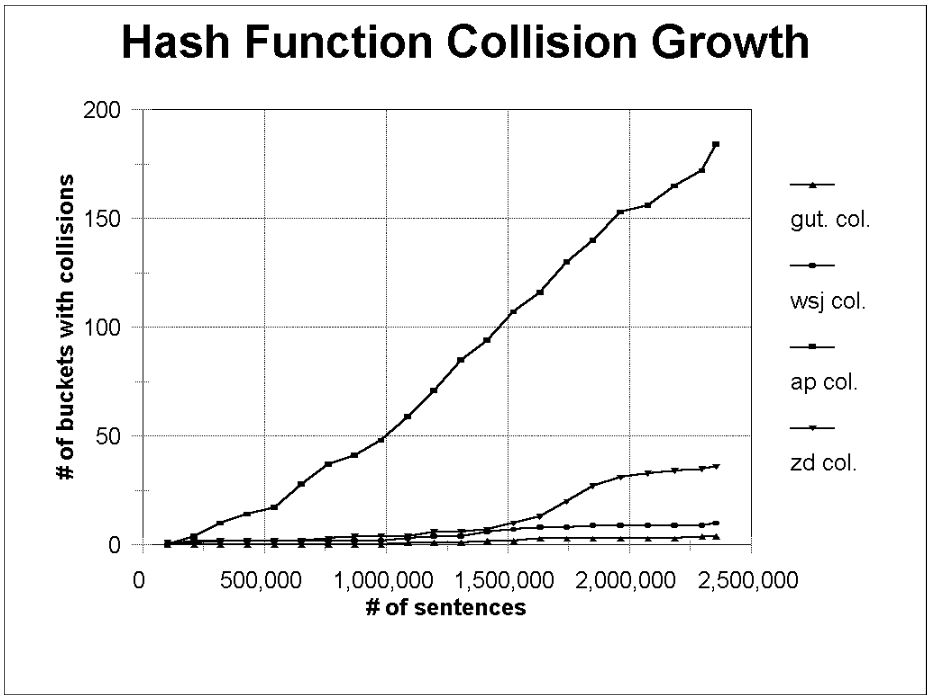

Figure 2 Growth in the number of collisions using the final hash function (Hash 5) over the four document collections.

Table 1 Collision behavior of Hash 5 applied to the four document collections.

Collection	# of hashable sentences	# of collisions
gutenberg	4,113,089	**21**
wsj	2,911,293	**15**
ap	3,575,343	**358**
zd	2,356,795	**36**

6. Copy detection

For our work, we have defined a measure for similarity that measures and reports the amount of overlap between two documents independently of the application in question. In other words, rather than measure our data according to specific results for which we are looking (i.e. plagiarism or subset), we measure the relatedness of two documents strictly according to the amount of similarity that exists between them. We then determine the suitability of two related documents to a given application through analysis of their similarity score and overlap distribution.

6.1. Measuring Similarity

Our similarity definition is as follows: given two documents d_1 and d_2, let $|d_1 \cap d_2|$ denote the number of sentences common to both d_1 and d_2. Let $|d_1|$ denote the number of sentences in document d_1, and let $|d_2|$ denote the number of sentences in document d_2. We define $sim(d_1,d_2)$, the similarity that exists between documents d_1 and d_2, to be the ordered pair

$$sim(d_1,d_2) = \left\langle \frac{|d_1 \cap d_2|}{|d_1|}, \frac{|d_1 \cap d_2|}{|d_2|} \right\rangle$$

The first term computes the number of sentences common to d_1 and d_2 divided by the number of sentences in d_1. It represents the fraction of d_1 contained in d_2. The second term computes the number of sentences common to d_1 and d_2 divided by the number of sentences in d_2. It represents the fraction of d_2 contained in d_1. We alternatively refer to the ordered pairs produced by $sim(d_1,d_2)$ as the similarity score of d_1 and d_2, and we refer to the individual values in the ordered pair as terms.

As an example, consider two documents d_1 (120 sentences) and d_2 (160 sentences) such that d_1 and d_2 have 80 sentences in common. $sim(d_1,d_2)$ returns <0.667, 0.500> as the similarity measure between d_1 and d_2, indicating that two-thirds of the sentences in d_1 are found in d_2 and half the sentences in d_2 are found in d_1. Note that $sim(d_1,d_2)$ measures the subset relationship between d_1 and d_2.

This correspondence to the subset relationship explains why the similarity measure returns a pair instead of a single number. Knowing that document d_1 contains sentences in document d_2 does not tell us about d_2's relationship to d_1–we need information about d_1's relationship to d_2 and d_2's relationship to d_1. Dropping either term or combining the terms into a single number results in a loss of information.

We give three reasons that our similarity measure is intuitive. First, the terms in the ordered pair are properly scaled. Identical documents are 100% similar (in both directions), and documents with no sentences in common report a similarity of <0.00, 0.00>. Partial overlap is reflected by an ordered pair that contains the amount of overlap that occurs between the documents.

Second, our similarity measure is easily computable. For any pair of documents d_1 and d_2, the user can verify the values in the ordered pair returned by the measure simply by counting the number of sentences common to both d_1 and d_2, and dividing this count by $|d_1|$ and $|d_2|$, respectively.

Third, our definition of similarity has a direct correspondence to the well-understood concept of subset. The definition and "meaning" of our similarity definition is intuitive and well-understood to the same extent that the notion of a subset between two sets is intuitive and well-understood.

6.2. Location of Overlap

The copy detection system draws on the sentence-based structure of a document to report the overlap distribution–the location within a document in which overlap occurs. An overlap distribution is generated for each of the two documents involved in the comparison.

A bit vector B tracks the overlap of each document D. ($B[i]$ corresponds to the i^{th} sentence in D.)

Determining the location of overlap between two documents d_1 and d_2 proceeds as follows: let document d_1 have bit vector B_1 and document d_2 have bit vector B_2. Let S be the set of sentences common to d_1 and d_2. Then, for each sentence $s \in S$, let i_s denote the position of s in d_1 and j_s denote the position of s in d_2. To record the location of overlap in document d_1 and d_2, set both $B_1[i_s]=1$ and $B_2[j_s]=1$.

The data in the bit vector contains the information for displaying the overlap distribution. We have implemented two types of output. Our first output is simply the bit vector itself: the character "1" is output if the entry has value 1, and the character "0" is output if the entry has value 0. For small documents, this first approach works well, but for large documents the output becomes long and difficult to read.

Our second output is for large documents: it partitions the bit vector into k-length clusters of contiguous entries (k, the *granularity*, denotes the number of entries in a cluster). During output, a single number is displayed for each cluster (the number of bit vector entries within the cluster that have the value 1).

7. System architecture

In this section, we briefly discuss the document registration, copy detection, and database components of our component-based system, depicted in Figure 3.

1. Document Registration Component. Given a document for input, the document registration component divides the document into its component sentences, hashes each sentence, and inserts each hash value (along with other information) into a database. The execution time of this procedure is dominated by the hashing involved. On a 350 MHz Intel Pentium II processor running Solaris x86, the unoptimized hash function (written in PERL) hashed 4.1 million sentences in 12 hours, which translates to roughly 100 sentences per second.
2. Copy Detection Component. To find the overlap between a test document and the database of registered documents, the copy detection system divides the test document into its component sentences and hashes each sentence. We then apply algorithms to quickly compare each hash value of the test document to the database. Each matching hash value in the database is retrieved, sorted, and grouped according to its document id. The output data is a visual distribution of the matches, the similarity score, and a list of the matching sentences.
3. Database Component. The database component contains several tables and operations necessary to support two types of queries on a hash value X:
   1. Return all (document id, sentence-based offset relative to the document) pairs whose hash value is X.
   2. Return all sentences whose hash value is X.

   The tables are indexed and stored in a manner that provides fast, efficient access. Unlike the other components, the database component is not directly visible to the user. The two queries above are posed by the other components during processing.

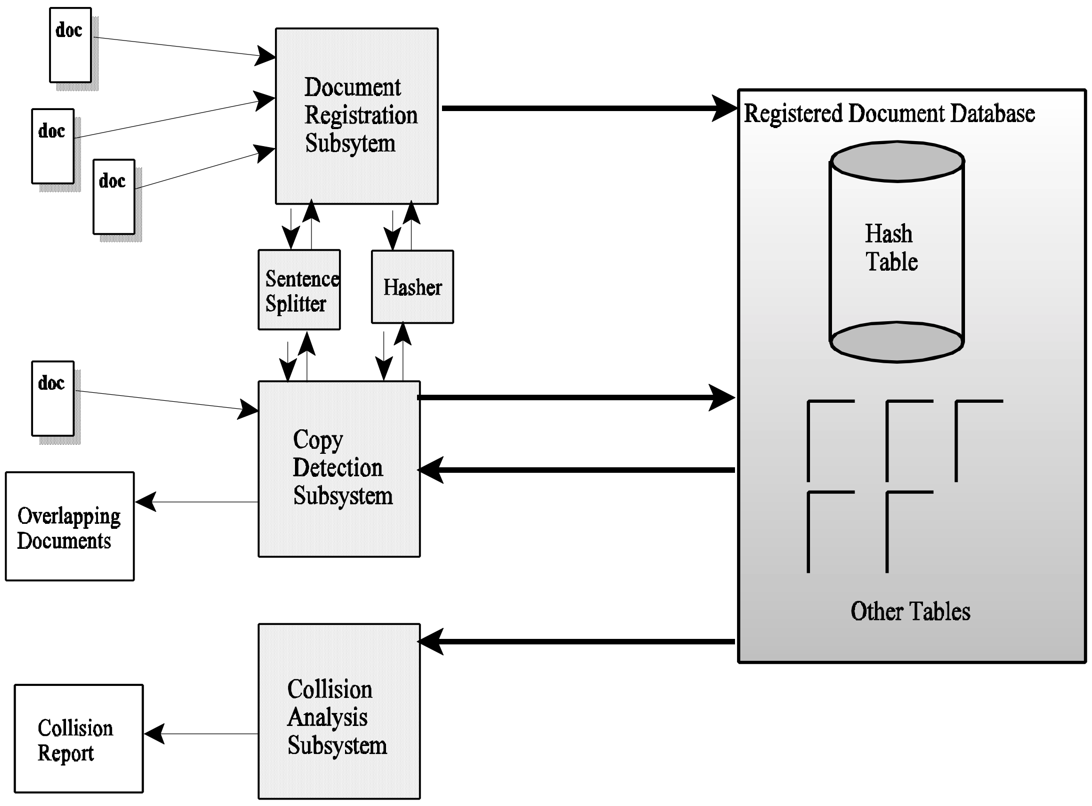

Figure 3 High level architecture of the copy detection system

8. Experiments

In this section, we report on three of the many performed experiments [22].

8.1. Experiment 1

On April 15, 1999, a man entered the Mormon Church's Family History Library in Salt Lake City, began firing randomly, and killed three people. That evening we retrieved from various web sites seven news articles that reported the shooting. We entered the articles in our copy detection system and compared one of the articles, which we named *abcnews.txt*, with the other six documents. Table 2 shows the results of these comparisons (with granularity of k=5). Each of the pairwise comparisons along with their similarity scores and overlap distributions are displayed, showing the portions of the documents that are found to be identical.

This experiment helped us understand our hash function's performance on the *ap* collection, which contains several closely related newspaper articles.

8.2. Experiment 2

Let D_1 be the concatenation of the entire Federalist Papers (7570 sentences); let D_2 be Federalist Paper #51 (76 sentences), taken from a different source. Table 3 vividly demonstrates our copy detection system's robustness in the face of documents whose sizes differ by a factor of 100.

8.3. Experiment 3

Research i is the i^{th} version of a research paper, i=1 to i=8. Table 4 shows the results of comparing successive versions of the paper. The similarity scores and overlap distributions illuminate the evolution of the successive drafts. Table 4 illuminates both where revisions occur and the density of the revisions.

9. Future work

We have identified six areas for future work:

- Currently, Hash 5's footprint is 14 bytes per sentence. We can add straightforward compression techniques to

Table 2 Comparison of the file *abcnews.txt* with registered documents covering the same story. The overlap granularity is set to k=5.

Document Pairs	# of matching sentences	Similarity Score	Overlap Distribution
abcnews.txt (test document)	30	<1.00, 1.00>	|5|5|5|5|5|5|
abcnews.txt			|5|5|5|5|5|5|
abcnews-2.txt	25	<0.74, 0.83>	|0|4|4|4|4|5|4|
abcnews.txt			|4|4|3|4|5|5|
cbsnews.txt	4	<0.13, 0.13>	|0|0|0|2|1|1|
abcnews.txt			|0|0|2|1|0|1|
cnn.txt	20	<0.65, 0.67>	|0|3|4|3|5|5|0|
abcnews.txt			|0|4|4|3|5|4|
deseretnews.txt	2	<0.03, 0.07>	|0|0|0|1|1|0|0|0|0|0|0|0|0|0|0|0|
abcnews.txt			|1|1|0|0|0|0|
msnbc.txt	17	<0.49, 0.57>	|3|3|3|0|2|2|4|
abcnews.txt			|3|3|3|4|2|2|
saltlaketribune.txt	19	<0.61, 0.63>	|2|3|4|2|4|3|1|
abcnews.txt			|2|4|3|3|5|2|

the hash function to reduce the footprint by 35% (to 9 bytes per sentence). How much further can we compress the hash values, and how does this affect the performance of the system?

- System processing time is dominated by the document registration process in which documents are decomposed into sentences, hashed, and placed in a database. Our platform can perform registration at a rate of 100 sentences/second. What can we do to increase this rate?
- Currently, our hash function is designed to group identical sentences, not just similar ones. A hash function that can properly group similar sentences would provide for broader comparisons between documents to be made. Such a similarity-based hash function should accept a parameter to control the amount of similarity that is desired.
- Our analysis of our hash function is largely empirical. Future work includes developing a theoretical model for Hash 5 that can predict, among other things, its scalability limits.
- The current implementation of our system is text-based. How should our line-oriented overlap distribution display be extended to a GUI to assist human evaluation of overlap significance? ([7] has given us some initial ideas.)
- Our current system is oriented towards sentence-based documents. However, many documents (including web documents) are not sentence-based in that they contain more phrases and sentence fragments than full sentences. Our system must be extended to detect overlap of non-sentence data.

10. Conclusion

This paper presented our sentence-based copy detection system for digital documents and discussed five contributions:

- an intuitive definition of similarity between documents;
- the ability to display the location of overlap between documents;
- the creation of a hash function which is robust over documents of different genres;
- the creation of a hash function which averages only tens of false positives for millions of sentences;
- the creation of a hash function which scales (at least) to millions of sentences.

Table 3 Results of the comparison between documents of widely varying sizes. The large document contains all of the Federalist Papers; the small document contains just one of the Federalist Papers. The distribution granularity is set to *k*=20.

Document Pairs	# of matching sentences	Similarity Score	Overlap Distribution
Federalist Paper #51 (small document)	72	<0.99, 0.01>	|19|20|18|15|
Combined Federalist Papers (large doc.)			|0|0...0|17|20|18|18|0|0...0|0|*

*With a granularity *k*=20, the 7570 sentences in the large document requires several lines to display. In the table, the overlap is abbreviated to show only the clusters in which overlap occurs. The actual overlap distribution is as follows:
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|**17|20|18|18**|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|

Table 4 Pair-wise comparison of successive drafts of the research paper, with the last row containing a comparison between the first draft and the final revision. The granularity is set to *k*=20.

Document Pairs	# of matching sentences	Similarity Score	Overlap Distribution
research-1.txt	133	<0.93, 0.58>	|20|18|15|20|19|20|19|02|
research-2.txt			|20|16|15|19|19|20|18|06|00|00|00|00|
research-2.txt	223	<0.98, 0.82>	|17|20|20|20|19|20|20|20|20|19|20|08|
research-3.txt			|11|20|20|20|19|20|20|20|20|18|00|02|20|13|
research-3.txt	177	<0.65, 0.60>	|11|12|10|10|13|08|14|13|16|12|16|17|15|10|
research-4.txt			|12|10|09|08|14|07|09|14|13|16|11|16|17|14|07|
research-4.txt	260	<0.88, 0.87>	|20|20|20|20|20|20|20|20|20|20|07|13|13|14|13|
research-5.txt			|20|20|20|20|20|20|20|20|20|20|07|11|15|09|18|
research-5.txt	296	<0.99, 0.99>	|19|19|19|20|20|20|20|20|20|20|20|19|20|20|20|
research-6.txt			|19|19|19|20|20|20|20|20|20|20|20|19|20|20|20|
research-6.txt	64	<0.21, 0.24>	|14|09|06|07|03|06|00|00|03|03|00|01|05|01|06|
research-7.txt			|14|08|07|07|04|05|00|01|05|00|03|03|04|03|
research-7.txt	251	<0.95, 0.95>	|20|20|20|20|20|20|15|17|19|20|17|19|20|04|
research-8.txt			|20|20|20|20|20|20|16|17|19|20|17|19|20|03|
research-1.txt	14	<0.10, 0.05>	|04|05|00|01|04|00|00|00|
research-8.txt			|04|04|01|01|02|02|00|00|00|00|00|00|00|00|

We generated four large test sets drawn from different types of data sources, and we observed good results when we applied the test sets to our system. At a high level, we described the architecture of the system. Finally, we reported the results of several experiments that illustrate the behavior, robustness, and scalability of our system.

11. References

[1] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman. Document marking and identification using both line and word shifting, Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies,

Boston, Massachusetts, April 1995.

[2] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman. Electronic marking and identification techniques to discourage document copying, IEEE Journal on Selected Areas in Communications 13(8): 1495-1504, October 1995.

[3] Sergey Brin, James Davis, and Hector Garcia-Molina. Copy detection mechanisms for digital documents, Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 1995.

[4] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web, Proceeding of the Sixth International World Wide Web Conference, April 1997.

[5] Douglas M. Campbell. A sentence boundary recognizer for English sentences, unpublished work, 1997.

[6] H. Garcia-Molina, S. Ketchpel, N. Shivakumar. Safeguarding and charging for information on the Internet, Proceedings of International Conference on Data Engineering (ICDE'98), Orlando, Florida, February 1998.

[7] Marti A. Hearst. TileBars: visualization of term distribution information in full text information access, Proceedings of the ACM SIGCHI Conference Human Factors in Computing Systems (CHI), Denver, Colorado, 1995.

[8] Nevin Heintze. Scalable document fingerprinting, Proceedings of the Second USENIX Workshop on Electronic Commerce, Oakland, California, Nov. 1996.

[9] Brewster Kahle. Preserving the Internet, Scientific American, March 1997. http://www.sciam.com/0397issue/0397kahle.html.

[10] Udi Manber, Finding similar files in a large file system, USENIX Winter 1994 Technical Conferences, San Francisco, California, January 1994.

[11] B.J. McKenzie, R. Harries, and T. Bell. Selecting a hashing algorithm, Software-Practice and Experience 20(2): 209-224, February 1990.

[12] Fred Mintzer, Jeffrey Lotspiech, and Norishige Morimoto. Safeguarding digital library contents and users: digital watermarking, D-Lib Magazine: http://www.dlib.org/, December 1997.

[13] D. Palmer and M. Hearst. Adaptive sentence boundary disambiguation, Proceeding of the Conference on Applied Natural Language Processing, Stuttgart, Germany, October 1994.

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

[15] Narayanan Shivakumar and Hector Garcia-Molina. SCAM: A copy detection mechanism for digital documents, Proceedings of the Second International Conference in Theory and Practice of Digital Libraries (DL'95), Austin, Texas, June 1995.

[16] Narayanan Shivakumar and Hector Garcia-Molina. Building a scalable and accurate copy detection mechanism, Proceedings of the 1st ACM International Conference on Digital Libraries (DL '96), Bethesda, Maryland, March 1996.

[17] Narayanan Shivakumar and Hector Garcia-Molina. The SCAM approach to copy detection in digital libraries. D-lib Magazine: http://www.dlib.org/, November 1995.

[18] Narayanan Shivakumar and Hector Garcia-Molina. Finding near-replicas of documents on the web, Proceedings of the International Workshop on the Web and Databases (WebDB'98), Valencia, Spain, March 1998.

[19] H. Garcia-Molina, L. Gravano, and N. Shivakumar. DSCAM: Finding document copies across multiple databases, Proceedings of the 4th International Conference on Parallel and Distributed Information Systems (PDIS'96), Miami Beach, Florida, Dec. 1996.

[20] Randy Smith. Copy Detection Systems for Digital Documents, Master's Thesis, June 1999.

[21] Project Gutenberg homepage: http://www.promo.net/pg/.

[22] Text Retrieval Conference (TREC) homepage. http://trec.nist.gov/.

[23] Glatt Plagiarism Services homepage: http://www.plagiarism.com/.