Comparison of Overlap Detection Techniques

Krisztián Monostori[1], Raphael Finkel[2], Arkady Zaslavsky[1], Gábor Hodász[3], Máté Pataki[3]

[1] School of Computer Science and Software Engineering, Monash University, 900 Dandenong Rd, Caulfield East, VIC 3145, Australia
{Krisztian.Monostori,Arkady.Zaslavsky}@csse.monash.edu.au
[2] Computer Science, University of Kentucky, 773 Anderson Hall, Lexington, KY 40506-0046, USA
raphael@cs.uky.edu
[3] Department of Automation and Applied Informatics, Budapest University of Technology and Economic Sciences
1111 Budapest, Goldmann György tér 3. IV.em.433., Hungary,
s8043hod@hszk.bme.hu, pm205@hszk.bme.hu

Abstract. Easy access to the World Wide Web has raised concerns about copyright issues and plagiarism. It is easy to copy someone else's work and submit it as someone's own. This problem has been targeted by many systems, which use very similar approaches. These approaches are compared in this paper and suggestions are made when different strategies are more applicable than others. Some alternative approaches are proposed that perform better than previously presented methods. These previous methods share two common stages: chunking of documents and selection of representative chunks. We study both stages and also propose alternatives that are better in terms of accuracy and space requirement. The applications of these methods are not limited to plagiarism detection but may target other copy-detection problems. We also propose a third stage to be applied in the comparison that uses suffix trees and suffix vectors to identify the overlapping chunks.

1 Introduction

Digital libraries and semi-structured text collections provide vast amounts of digitised information online. Preventing these documents from unauthorised copying and redistribution is a hard and challenging task, which often results in avoiding putting valuable documents online [5]. The problem may be targeted in two fundamentally different ways: copy prevention tries to hinder the redistribution of documents by distributing information on a separate disk, using special hardware or active documents [6] while copy detection allows for free distribution of documents and tries to find partial or full copies of documents.

One of the most current areas of copy detection applications is detecting plagiarism. With the enormous growth of the information available on the Internet, students have a handy tool for writing research papers. With the numerous search

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 51–60, 2002.

engines available, students can easily find relevant articles and papers for their research. But this tool is a two-edged sword. These documents are available in electronic form, which makes plagiarism feasible by cut-and-paste or drag-and-drop operations.

Academic organisations as well as research institutions are looking for a powerful tool for detecting plagiarism. There are well-known reported cases of plagiarised assignment papers [1,2]. Garcia-Molina [5] reports on a mysterious Mr. X who has submitted papers to different conferences that were absolutely not his work. Such a tool is also useful for bona fide researchers and students who want to be sure not to be accused of plagiarism before submitting their papers.

Copy-detection mechanisms may be used in many other scenarios. Nowadays books are published with companion CDs containing the full-text version of the book, which is very useful for searching purposes but bears the risk that the book may be posted on Web sites or newsgroups. A copy-detection tool could use the full-text version of the book and try to locate near replicas of the document on the Internet. Search-engines could also benefit from copy-detection methods: when search results are presented to the user they could be used as a filter on documents, thus preventing users from e.g. reading the same document from different mirror sites. There are various other application areas where copy-detection mechanisms could be used, but they are not listed here because of space limitations. We believe that copy-prevention mechanisms are not the right solution because they impede bona fide researchers while protecting against cheating and law infringement.

In this paper, we compare existing copy-detection mechanisms, which, we have found, have a lot in common. We suggest different application areas for different methods. The most common two-stage approach is discussed in the following section. Section 3 discusses different alternative approaches to split the document into chunks based on our test results. Section 4 presents alternative approaches to select a representative set of chunks, and in Section 5 we summarize our results and look at future work.

2 Overview of Copy-Detection Methods

There are commercial copy-detection systems on the market [10, 4], whose algorithms are not published for obvious reasons. Research prototype systems include the SCAM (Stanford Copy Analysis Method) system [5], the Koala system [7], and the “shingling approach” of [3].

The common approach of these prototype systems to the copy-detection problem can be summarized in five steps:

1. Partition each file into contiguous chunks of tokens.
2. Retain a relatively small number of representative chunks.
3. Digest each retained chunk into a short byte string. We call the set of byte strings derived from a single file its **signature** or **fingerprint**.
4. Store the resulting byte strings in a hash table along with identifying information.

5. If two files share byte strings in their signatures, they are **related**. The closeness of relation is the proportion of shared byte strings.

We defer the discussion of the first two stages to Sections 3 and 4, and we discuss steps 3,4, and 5 in the following subsections.

2.1 Digesting

We could store the chunks themselves, but we choose not to do so for two reasons. First, chunks may be quite large, and we wish to limit the amount of storage required. Second, chunks contain the intellectual property of the author of the file that we are processing. We prefer not to store such property in order to reduce fears that our tools can themselves be used to promote plagiarism or that the database can be used for breaches of confidentiality and privacy.

Instead of storing the chunks, we reduce them by applying a digesting tool. We use the MD5 algorithm [11], which converts arbitrary byte streams to 128-bit numbers. Storing 128 bits would waste too much storage without offering any significant advantage over shorter representations. Of course, the more hex bits we store, the more accurate our system will be. Storing fewer bytes means that two chunks may produce the same digest value even when they are different. These undesired cases are called false positives. In Section 3 where we discuss different chunking strategies, we analyse the effect of different digest sizes on the accuracy of the system. Here we also note that in our system we use an extra stage when we compare documents. This extra stage uses exact string matching techniques to find the actual overlapping chunks. Thus some false positives are not of great concern, because they are eliminated in this stage. We refer to this extra stage as the MDR approach [8, 9].

Of course, other systems use other hashing schemes to create the digest representation of a given chunk, and it would be interesting to compare the effect of different hashing schemes on the number of false positives, but that comparison is beyond the scope of this paper.

2.2 Storing Hash Values

Hash values generated using the methods described above need to be stored in some kind of a database. We may choose to store the hash values in a general-purpose database management system, or we can develop a special-purpose system tailored to store the hash values and their postings efficiently. In our system we have used Perl [12] hash data structures to store hash values.

2.3 Comparison

Different similarity measures can be defined for different application areas. Here we list the three most common measures. Let us denote the digest of file F by d(F).

- Asymmetric similarity $$a(F,G) = \frac{|d(F) \cap d(G)|}{|d(F)|}$$
- Symmetric similarity $$s(F,G) = \frac{|d(F) \cap d(G)|}{|d(F)| + |d(G)|}$$
- Global similarity $$g(F) = \frac{|d(F) \cap (\cup_G d(G))|}{|d(F)|}$$

Of course, the size of documents has a significant effect on the results. For example if document F is much smaller than document G, and document F is entirely contained in document G, then the *a*(F,G) value will be much higher than the *s*(F,G) value. The global similarity measure is used for defining the similarity between a given document and a set of documents.

3 Chunking

In this section we analyse different chunking methods proposed in different prototype systems. Subsection 3.1 discusses existing techniques while subsection 3.2 analyses our test results and proposes a new technique.

3.1 Existing Chunking Methods

Before we compare chunking methods, we have to decide what is the smallest unit of text that we consider overlap. One extreme is one letter, but in this case, almost all English language documents would have 100% overlap with other documents, because they share the same alphabet. The other extreme is considering the whole document as one unit. This method could not identify partial overlap between documents.

Prototype systems and our experiments show that the length of chunks should be somewhere between 40 and 60 characters for plagiarism-detection purposes. The Koala system [7] uses 20 consonants, which translates into 30-45 characters; the "shingling approach" [3] considers 10 consecutive words, which is approximately 50-60 characters; the SCAM system [6] analyses the effect of different chunk sizes: one word, five words, ten words, sentences. Our exact-comparison algorithm (MDR approach) used in the extra stage of the comparison process uses 60 characters as a threshold.

The selection of chunking strategy has a significant effect on the accuracy of the system. In the ideal case, we would index all possible α-length chunks of a document. In a document of length *l* there are *l-α-1* possible α-length chunks [7]. A suffix tree is a data structure that stores all possible suffixes of a given string, thus all possible α-length chunks. Suffix trees are used in the MDR approach.

We could consider non-overlapping chunks of α-length character sequences. The problem with this approach is that adding an extra word or character to the document would shift all boundaries, so two documents differing only in a word would produce two totally different sets of chunks.

The "shingling approach" of Broder et al. [3] uses words as the building blocks of chunks. Every ten-word chunk is considered. As an example, consider the following sentence: "copy-detection methods use some kind of a hash-function to reduce space requirements".

Ten-word chunks generated from this "document": "copy-detection methods use some kind of a hash-function to reduce", "methods use some kind of a hash-function to reduce space", and "use some kind of a hash-function to reduce space requirements".

They call these ten-word chunks shingles. To store every ten-word chunk requires too much space, so Broder et al. [3] also consider a strategy to select representative chunks of a document. Selection algorithms are discussed in the next section.

Garcia-Molina et al. [6] compare different chunking strategies. At the finest granularity, word chunking is considered with some enhancements, such as eliminating stopwords. Another possibility is to use sentences as chunks to be indexed. The problem with sentences is that sentence boundaries are not always easy to detect. They tested their algorithm on informal texts, such as discussion-group documents, because of their informality they often lack punctuation. Sentence chunking also suffers when identifying partial sentence overlap. We have run sentence chunking on our document set of RFC documents and found that the average chunk length using only the full stop as a termination symbol is about 13 words; if we consider other punctuations as chunk boundaries the average is 3.5 words. The problem with sentence chunking is that it does not allow for tuning. By tuning we mean that different problem areas might require different chunk lengths.

To overcome the boundary-shifting problem of non-overlapping chunks, Garcia-Molina et al. [6] introduced hashed-breakpoint chunking. Instead of strict positional criteria to determine chunk boundaries, such as every 10th word is a chunk boundary, they calculate a hash value on each word and whenever this hash value modulo k is 0, it is taken as a chunk boundary. They studied the performance of their prototype system with $k=10$ and $k=5$. The expected length of a chunk with this hashed-breakpoint chunking is k words. As discussed earlier, $k=10$ is closer to the empirically defined threshold value of 40-60 characters. The problem with this approach is that the chunk sizes may differ. We can easily have one-word chunks and also 15-20 word chunks. See the next subsection about this problem.

3.2 Chunking Strategy Tests

The most promising chunking strategy is hashed-breakpoint chunking. It avoids the shifting problem without the need to store overlapping chunks. In this subsection we analyse the hashed-breakpoint strategy in details.

It is true that the expected chunk length is *k* in case of *k*-hashed breakpoint chunking, but if a common word happens to be a chunk boundary, the average chunk length may be much smaller than the expected average.

We have used two document sets for comparison. The first set is the set of RFC (Request for Comment) documents; the second set comprises different Hungarian translations of the Bible. We have chosen these two sets because we expect some overlap within these document sets for obvious reasons.

Figure 1 shows the average chunk length in the function of the *k* value. We can see that the higher the *k* value, the higher the chance is of greater deviation from the expected result. This behaviour can be explained by the uncertainty of this chunking strategy.

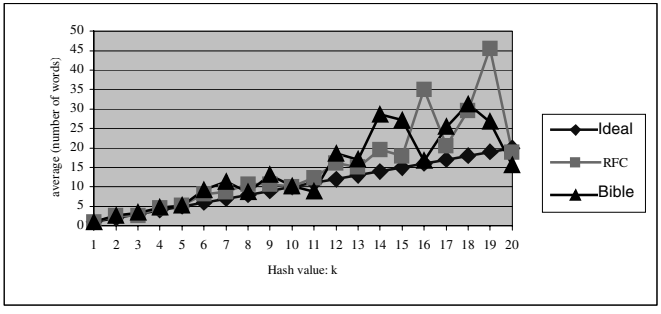

Fig 1. Average chunk length

We have picked nine pairs of documents from the Bible translations and compared them with different hash values. The results shown in *Figure 2* reflect the uncertainty of this chunking method.

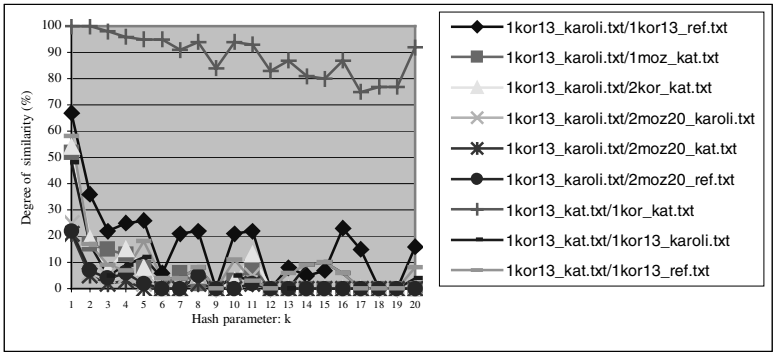

Fig 2. Overlap percentage

The chosen pairs are different translations of the same part of the Bible. The correlation between *Figures 1* and *2* is obvious. For example, the first pair has a peak at *k*=16 following a low at *k*=14 and *k*=15. In *Figure 1*, we can see that at *k*=14 and *k*=15 the average chunk length is higher than expected; if we have longer chunks,

the chance for overlap is smaller. On the contrary, at k=16 we have a low in *Figure 1*, which means that we have shorter chunks, so the chance for overlap is higher.

In *Figure 2* we can also see that the chosen k value has a significant effect on the amount of overlap detected. We propose to use more than one k value for comparison, and we can either choose the average of the reported overlaps or the maximum/minimum values depending on the actual application. This strategy eliminates the sensitivity of the algorithm to different k values. *Figure 3* shows the results of this approach. We aggregated the number of chunks reported by different k values and calculated the result based on the total number of chunks.

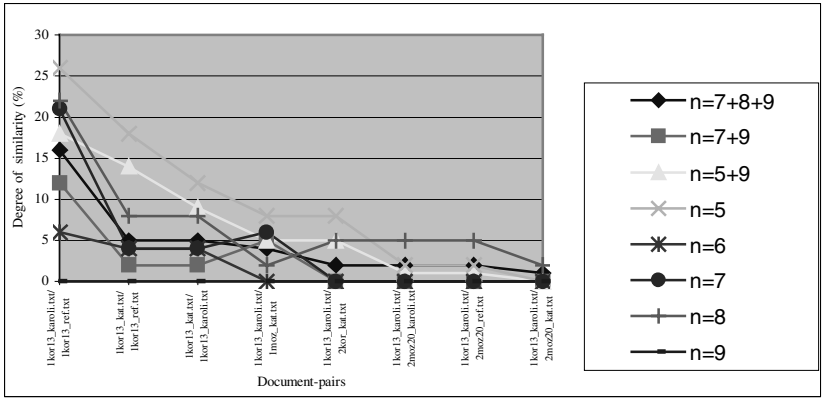

Fig 3. The effect of aggregate k values

Of course, in our applications false positives are more desirable than false negatives, because false positives can be eliminated in our extra stage of exact comparison, while we will never report documents missed by the first stage.

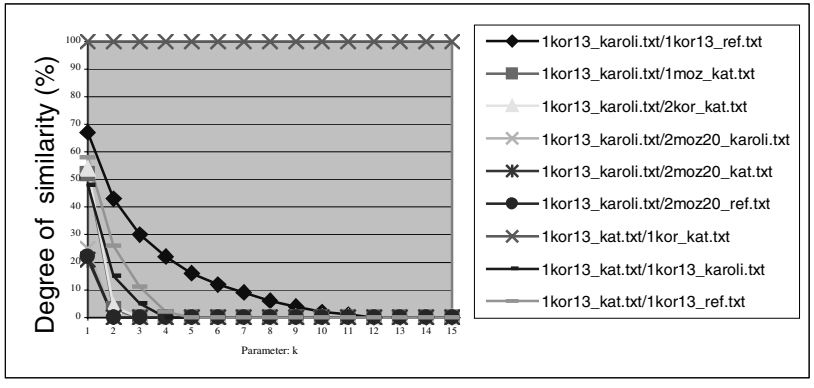

Fig. 4. Overlapping chunks

We have also conducted a test on different chunk sizes. In this test we have used overlapping chunks of different sizes because of their better predictability. The results are shown in *Figure 4*. As we expected longer chunks result in less detected overlap. Based on these experiences, we can conclude that for finding documents similar in style, we can choose a k value of 2 or 3, which would find similar phrases. For plagiarism detection, k=8 or k=9 seems to be a good parameter value, while for detecting longer overlaps, we can choose values greater than 10.

False positives are a problem common to every approach. We use hashing to identify chunks, and we may have identical hash values when the original chunks do not match. *Table 1* contains the number of false positives for different chunking methods and bit depths. These hash values have been generated by the MD5 algorithm by keeping only the left-most k bits.

Table 1. False positives

Method	bit-depth	false positives	false positive (%)
hashed breakpoint (k=6)	24	8434	1.6868
hashed breakpoint (k=9)	24	6790	1.3580
overlapping (k=6)	24	7115	1.4230
sentence	24	13954	2.7908
hashed breakpoint (k=6)	32	23	0.0046
hashed breakpoint (k=9)	32	21	0.0042
overlapping (k=6)	32	26	0.0052
sentence	32	15	0.0030

The tests were carried out on 500,000 chunks. The results show a dramatic decrease in the number of false positives when we move from 24 bits to 32 bits. We suggest using 32 bits, not only because the number of false positives is less than 0.01%, but it is also an easier data width to handle in today's computers.

4 Fingerprint Selection

In the second stage of comparison we could store all chunks, but long files lead to many chunks. Dealing with them all uses space for storing them and time for comparing them against other stored chunks.

However, it is not necessary to store all chunks. Other systems discussed in Section 2 are using some kind of a fingerprinting or culling strategy to select representative chunks. The strategies used in those systems are mainly random. In the following subsection, we propose a more strategic approach and we support the applicability of our strategy by test results in subsection 4.2.

4.1 Selection strategies

A short chunk is not very representative of a text. The fact that two files share a short chunk does not lead us to suspect that they share ancestry. In contrast, very long chunks are very representative, but unless a plagiariser is quite lazy, it is unlikely that a copy will retain a long section of text.

We therefore discard the longest and the shortest chunks. We wish to retain similar chunks for any file. We have experimented with two culling methods. Let n be the number of chunks, m the median chunk size (measured in tokens), s the standard deviation of chunk size, b a constant, and L the length of an arbitrary chunk.

Sqrt. Retain $\lceil \sqrt{n} \rceil$ chunks whose lengths L are closest to m.

Variance. Retain those chunks such that $|L-m| \leq bs$. Increase b, if necessary, until at least $\sqrt{n}$ chunks are selected. We start with b=0.1.

4.2 Test Results

We have tested our fingerprinting method on the RFC data set. We have found that the Sqrt method does not store enough chunks, thus the Variance method is preferable. In *Table 2* we show the results of these tests based on some RFC documents with known overlap. The table shows asymmetric similarity, that is RFC 1 compared to RFC 2 does not necessarily provides the same result as RFC 2 compared to RFC 1. We consider the MDR method as accurate because it is based on exact matching. SE is our signature extraction method while OV is the overlapping chunk method.

Table 2 shows that our SE method tends to underestimate large overlaps while overestimating small overlaps (overlap is given as percentage). The overlapping-chunks method seems to provide more accurate results but at a much higher storage cost. Overestimation is not a problem in our system because we use the MDR approach as a final filter, which correctly identifies overlapping chunks.

Table 2. Asymmetric similarities

RFC 1	RFC 2	MDR 1	MDR 2	SE 1	SE 2	OV 1	OV 2
1596	1604	99	99	91	92	94	94
2264	2274	99	99	96	95	94	94
1138	1148	96	95	93	92	91	89
1065	1155	96	91	71	68	84	79
1084	1395	86	84	58	64	79	75
1600	1410	72	77	52	48	58	61
2497	2394	19	17	33	27	16	15
2422	2276	18	3	23	6	15	2
2392	2541	16	12	27	17	13	10

5 Conclusion and Future Work

In this paper we have analysed different chunking strategies used in copy-detection systems. We know of no other studies that have analysed these systems from this aspect. We have made suggestions on how to select chunking strategies and also touched on the problem of fingerprint selection, which is also crucial because of space-efficiency. Our SE algorithm provides very close results to the exact comparison and it overestimates small overlaps, which is a more desirable behaviour than underestimating, because false positives can be found by applying an extra filter on the results. Of course, too many false positives are not desirable, either. In the future we will also study the effects of different hash functions on false positives. We plan to run our tests on more document sets and we are also developing an online system where these methods are available for copy-detection applications (more specifically plagiarism detection).

6 References

1. Argetsinger A. Technology exposes cheating at U-Va. *The Washington Post*, May 8, 2001.
2. Benjaminson A. Internet offers new path to plagiarism, UC-Berkeley officials say. *Daily Californian*, October 6, 1999.
3. Broder A.Z., Glassman S.C., Manasse M.S. Syntatic Clustering of the Web. *Sixth International Web Conference*, Santa Clara, California USA. URL http://decweb.ethz.ch/WWW6/Technical/Paper205/paper205.html
4. EVE Plagiarism Detection System. *URL http://www.canexus.com*, 2000
5. Garcia-Molina H., Shivakumar N. The SCAM Approach To Copy Detection in Digital Libraries. *D-lib Magazine*, November, 1995.
6. Garcia-Molina H., Shivakumar N. Building a Scalable and Accurate Copy Detection Mechanism. *Proceedings of 1st ACM International Conference on Digital Libraries (DL'96)* March, Bethesda Maryland, 1996.
7. Heintze N. Scalable Document Fingerprinting. *Proceedings of the Second USENIX Workshop on Electronic Commerce*, Oakland, California, 18-21 November, 1996. URL http://www.cs.cmu.edu/afs/cs/user/nch/www/koala/main.html
8. Monostori K., Zaslavsky A., Schmidt H. MatchDetectReveal: Finding Overlapping and Similar Digital Documents. *Information Resources Management Association International Conference* (IRMA2000), 21-24 May, 2000 at Anchorage Hilton Hotel, Anchorage, Alaska, USA. pp 955-957, 2000.
9. Monostori K., Zaslavsky A., Schmidt H. Parallel Overlap and Similarity Detection in Semi-Structured Document Collections. *Proceedings of 6th Annual Australasian Conference on Parallel And Real-Time Systems (PART '99)*, Melbourne, Australia, 1999. pp 92-103, 1999.
10. Plagiarism.org, the Internet plagiarism detection service for authors & education. *URL http://www.plagiarism.org* , 1999.
11. Rivest R. L.. RFC 1321: The MD5 Message-Digest Algorithm. *Internet Activities Board*, April 1992.
12. Wall L. and Schwartz R. L. Programming Perl. *O'Reilly & Associates*, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1992.