Finding Plagiarism by Evaluating Document Similarities

Jan Kasprzak, Michal Brandejs, and Miroslav Křipač
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
kas@fi.muni.cz, brandejs@fi.muni.cz, kripac@fi.muni.cz

Abstract: In this paper we discuss the approach we have used for finding plagiarized passages of text during the PAN'09 plagiarism detection competition. We describe the existing anti-plagiarism system we use in the Czech National Archive of Graduate Theses. We then discuss the modifications to this system which have been necessary in order to fit the results to the competition rules. We also present a performance data of the described system, and the possible improvement for our production systems, which result from the code written for the PAN'09 competition.
Keywords: Plagiarism, Similar Documents, Document Overlap, Distributed Computing, Parallelism

1 Background

At Masaryk University, the study administration is being supported by the Masaryk University Information System (IS MU, 1999–2009). The integral part of this system is the distributed document storage, used by various subsystems, including e-learning agendas, archive of graduate theses, etc. The document storage is a feature-rich subsystem with some properties similar to the common file systems (hierarchical directory-based storage, object as a stream of bytes, etc.).

Some features are quite unique to this document storage: the storage supports multiple versions of the same document (e. g. DOC, PDF, and plain text) as a single entity, automatic conversion between the file formats (including OCR of the scanned PDF files, generating thumbnail images for the picture files such as JPEG, etc.), distributed replication with strong checksums, wide set of access rights, etc.

Since August 2006, the storage subsystem supports also finding similarities between documents, in order to assist with discovering plagiarism. The first version of the system has been a prototype coded in Perl with the DBI interface, using Oracle database as a metadata back-end.

In early 2008, the system has been replaced by the custom distributed solution, outlined in our earlier work (Kasprzak et al., 2008), distributed aspects of which we further discuss in (Kasprzak, Brandejs, and Brandejsová, 2009). The same underlying system is currently in use also in the Czech National Archive of Graduate Theses (Theses.CZ, 2008–2009). This distributed system has been used as a basis for solving the external plagiarism task in the PAN'09 competition (PAN'09, 2009).

2 General Approach

There are several possible approaches for finding similarities in a given base of documents, some of them are discussed and compared in (Monostori et al., 2002). The IS MU and Theses.CZ anti-plagiarism system currently uses the approach similar to the one described by (Finkel et al., 2002).

2.1 Tokenization

We use words as the base units which we handle. In order to overtake the need of stemming (which would bring the dependence on the particular language to the system) and also to handle some means of obfuscation of the Czech language we translate the words to the US-ASCII by stripping off diacritics[1], and ignoring short words, which in the Czech language are mostly prepositions. We do not use stop words of any kind.

[1] For example: `tučňák` → `tucnak`

2.2 Creating the Index

The tokens (words) are then joined into *chunks*. We use overlapping sequences of several words, and we have generally had better results with shorter chunks of about 4 to 6 words than the larger chunks as used by (Broder, 1997) or (Finkel et al., 2002). The chunks are then hashed by a hash function (for our purposes, the value range of 28 to 32 bits seems to be sufficient, even though the probability of hash function collisions is higher with smaller number of bits). There are no special requirements to the hash function itself—we have used the hinghest n bits of the MD5 hash (Rivest, R., 1992).

The mapping of document ID to the sequence (or a set) of hash values is then constructed, and an *inverted index* mapping the hash value to the sequence of document IDs is computed from it.

2.3 Computing the Similarities

This inverted index is then used for finding similarities in a given document base. For now, the system computes only a single numeric value for each pair of documents in a given set. This value represents the number of chunks which these two documents have in common (not taking the possible hash function collisions into the account).

The most common use case is to discover which documents are similar to the given document (e. g. a newly imported thesis). We postpone the computation of the actual similar passages of the text to the time when the user wants to see them.

2.4 Hardware Configuration

The IS MU and Theses.CZ systems use a common document base of about 1,300,000 documents. The anti-plagiarism software runs on a cluster of 45 non-dedicated PCs with various dual-core CPUs (AMD and Intel), and a dedicated server with Oracle database for storing the computed results. Recomputing the similarities across the whole document base takes about three hours, most of which is spent by importing the results to the database. The incremental run (after adding some more documents to the system) then takes 12 to 25 minutes depending on the overall system load (the servers in the cluster have other tasks to do besides computing the similarities).

3 The PAN'09 Competition

The requirements for the PAN'09 competition were quite different to what we currently do in IS MU and Theses.CZ systems. The main difference was that we should not only find the similarity between the documents, but also to show exactly where the similarities are. Another important rule to consider was the granularity measure, which had been very strong, especially in the earlier version of the rules.

The first approach we wanted to try was simply to import both the suspicious documents and the source documents to the IS MU system, and let it find the similarities. This would have been a very straightforward solution, requiring no programming except post-processing the results on our side. Further examination of the data led us to the opinion that we can do better by modifying our software to match the requirements of the competition.

We have taken the core modules of our system and modified them to run on a single multi-core computer. Both the source corpus and the set of the suspicious documents were quite small (relative to what we have to handle in our production systems), so the relevant data could fit to the RAM of a single mid-range server.

3.1 Rich Tokenization

We have modified the tokenization process to get not only a list of words of a given document but also for each word to include the position of the word in the document expressed as two distinct numbers: firstly as the offset of the first character in the word from the beginning of the document, and secondly the count of the (non-ignored) words discovered so far in this document.

Using this additional data, we can construct the document chunks with the extra attributes: the offset of the first and last character of the chunk[2], and the sequence number of that chunk[3].

The tokenization process has been further modified to include all the Unicode alphanumeric characters as word characters. In our production systems, we do not count digits

[2] The offset of the last character in a chunk is computed from the offset of the last word of that chunk and the length of that word.

[3] The sequence number of that chunk is equal to the sequence number of the first word of that chunk.

as word characters, because a common case of plagiarism of student seminar works e. g. in biology is to take the work of the other student, and simply change the numbers in measurements.

3.2 Computing the Inverted Index

The computation of the inverted index (i. e. mapping the chunk hash value to the list of document IDs) has been modified to contain the additional data (the chunk sequence number, and range of characters which the chunk covers). In order to allow the case when a single plagiarized passage has more than one possible source passage in the same source document, we have also allowed the repeated occurences of the same chunk hash value within the same source document.

3.3 Finding the Similar Document Pairs

Using this enhanced inverted index, we can now tokenize and evaluate the suspicious documents in order to find similarities. For each suspicious document, we split it to the chunks, and look up their hash values in the index. This gives us the list of the documents, and positions of the chunks in them[4]. We can now use a cut-off value, and further handle the document pairs with at least this value of common chunks.

For the competition itself, we have used the cut-off value of 20 chunks, which together with 5-word chunks gives us a minimum of 24 common words that we consider a similarity between documents.

4 Discovering Similar Passages

In order to fulfill the requirements of the competition, we have not only to compute the similarity of the document pairs, but also to show exactly which passages are similar. The computation against an inverted index gives for each suspicious document the list of possible source documents, and for each source document a list of common chunks. With each common chunk we have its sequence number and characters range from the suspicious document, and (possibly more than one) sequence number and character range from the source document.

[4]Remember that a chunk from the suspicious document can be identical to several chunks in one source document.

Using this data, we can further narrow the scope, and keep only the larger similar passages. How to compute the similar continuous passages from the data is not clear. Our approach is to consider only "dense enough" intervals of the suspicious document, which also map to the "dense enough" intervals of the source document.

Since the computation has to be done both from the point of view of the suspicious document and the source document, for the algorithm it does not matter which document we are currently looking at. We will further use the terms D_1 and D_2.

We have considered only intervals of chunks of D_1 matching the following criteria:

1. The first and the last chunk of this interval are present in the D_2.
2. The interval should have at least 20 (possibly overlapping) chunks, which are also present in D_2.
3. Between each two adjacent chunks from the interval which are also present in D_2, there should be at most 49 chunks which have no matching chunk in D_2.

This interval we will hereby call a *valid interval*.

For example, when the suspicious document we have chunks numbered

$$50, 100, 150, ..., 950, 1000$$

which are all present in a particular source document, we consider the interval 50–1000 to be a valid interval.

We further process only those valid intervals, which also map to the valid interval in some source document. We consider part of the suspicious document covered by the chunks from this valid interval to be a plagiarized passage. The plagiarized passage can be computed using the algorithm, described in the in the next subsection:

4.1 Algorithm: Valid Intervals

The input of the algorithm is a list of pairs (chunks ID in D_1, matching chunk ID in D_2). If one chunk in D_1 maps to more than one chunk in D_2, the list has more than one entry for this chunk.

1. Set the local variable `depth` to 0.

2. Sort the list of pairs by the chunk ID in D_1.
3. Split the list to the largest possible valid intervals in D_1, ignore the chunk ID pairs which are not present in any valid interval.
4. If there is only one valid interval covering the whole input list, increase the `depth` variable by 1.
5. If the `depth` variable is equal to 2, return the whole range of chunk IDs as the resulting plagiarized passage.
6. For each valid interval, do the following:
   (a) Create a new list of chunk ID pairs as (chunk ID in D_2, chunk ID in D_1), where the chunk ID in D_1 is from the current valid interval.
   (b) Set the variable `depth` to 1.
   (c) Rerun recursively the algorithm, starting from the step 2.

4.2 Postprocessing

During the postprocessing phase, we remove possible overlapping passages for each suspicious document, keeping only the largest passage from the set of overlapping ones. This is to meet the nature of the competition data. In the real system, we would like to keep all the possible similarities, even the overlapping ones.

5 Practical Results

5.1 Implementation

The existing implementation of IS MU and Theses.CZ anti-plagiarism system uses a mixture of C and Perl code (C for performance-critical code like generating an inverted index of chunks or searching this index, Perl for feature-rich parts including the communication with the SQL database and generating chunks from the text files). It is portable to any 64-bit POSIX system. For the PAN'09 competition, it had to be only slightly modified, and a new valid interval evaluation and postprocessing system has been written. We have worked on the PAN'09 competition for four days, and additional half a day after the deadline has been extended. We did not do any fine-tuning against the evaluation formula, based on the development corpus.

5.2 Performance

The computation for the PAN'09 competition has been run on a single mid-range server: dual Xeon E5472 (3.0 GHz, total of 8 cores), 64 GB RAM, and a RAID-10 array of eight 15k RPM disks. The system runs Fedora Linux with the Ext4 filesystem. Most parts of the computation have been parallelized on a document-by-document or a document pair-by-pair basis.

Generating the inverted index from the corpus of 7124 source documents took 34 minutes. Finding the matching chunk pairs against the 7124 suspicious documents took about 38 minutes. Computing the maximum valid intervals from this data, postprocessing and generating the XML output files took 2 minutes. Because of the relatively big RAM available in the server, the computation was mostly CPU-bound.

5.3 The Benefit of Valid Intervals

Computing the valid intervals instead of just the similarity between the documents can surprisingly enough be a win not only from the viewpoint of getting a more meaningful data, but also from the performance standpoint: In our existing systems, most of the time during the computation is spent by inserting the similarity data to the database. Using the valid intervals can greatly reduce false-positives, and thus the number of rows (document pairs) which we need to insert to the database:

In the development corpus we had about 576,000 document pairs, which had at least 20 chunks in common. Actually looking at those chunks and keeping only those which form a valid interval both in the suspicious document and in the source document reduced this number to about 18,000 document pairs with 47,000 similar passages.

As it can be seen from the previous subsection, the cost of this step is relatively insignificant when compared to the other steps, even though we have implemented this step purely in Perl.

6 Conclusion

We have implemented the system which can find similar documents relatively fast, given a multi-core machine. The same approach can even be used on a cluster of computers, which can provide a significant benefit of a distributed memory for large document sets.

This system can detect even moderately obfuscated similar passages in a given document base. In the development corpus, we have found a big number of similar passages, which have not been annotated in the development data as plagiarized[5].

In the competition itself, we had the highest recall ratio[6] amongst all the teams (69.67 %, the second highest was 65.85 %). Given our relatively poor precision ratio (we were 7th in this parameter with the precision of 55.73 %, the highest precision reached was 74.73 %), we may even have found the biggest number of *all* the similar passages, including those not generated by the machine plagiarist.

Our system currently cannot handle translations (although the work is in progress to handle a plagiarism between Czech and Slovak languages, which are quite similar in structure and vocabulary). We also cannot handle highly obfuscated text, which to us non-native English speakers does not even look similar to the source text[7].

We have seen that evaluating the particular similar passages instead of just the overall document similarity can be a significant improvement, so this result from the competition is an enhancement to incorporate back to our production systems.

Acknowledgements

We would like to thank the organizers of the PAN'09 competition. Taking part in the competition has been a very enlightening experience for us.

References

Broder, A.Z. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29, Jun.

Finkel, Raphael A., Arkady Zaslavsky, Krisztián Monostori, and Heinz Schmidt. 2002. Signature extraction for overlap detection in documents. In *ACSC '02: Proceedings of the twenty-fifth Australasian conference on Computer science*, pages 59–64, Darlinghurst, Australia. Australian Computer Society, Inc.

IS MU. 1999–2009. Masaryk University Information System. `http://is.muni.cz/`.

Kasprzak, J., M. Brandejs, and J. Brandejsová. 2009. Distributed aspects of the system for discovering similar documents. In *ITA 09: Proceedings of the Third International Conference on Internet Technology and Applications*.

Kasprzak, J., M. Brandejs, M. Křipač, and P. Šmerk. 2008. Distributed system for discovering similar documents. In *ICEIS 2008: Proceedings of the Tenth International Conference on Enterprise Information Systems, Vol. DISI – Databases and Informations Systems Integration*, pages 437–440. INSTICC (Institute for Systems and Technologies of Information, Control and Communication), Setúbal, Portugal.

Monostori, Krisztián, Raphael A. Finkel, Arkady B. Zaslavsky, Gábor Hodász, and Máté Pataki. 2002. Comparison of overlap detection techniques. In *ICCS '02: Proceedings of the International Conference on Computational Science-Part I*, pages 51–60, London, UK. Springer-Verlag.

PAN'09. 2009. 1st International Competition on Plagiarism Detection. `http://www.webis.de/pan-09`.

Rivest, R. 1992. RFC1321: The MD5 Message-Digest Algorithm. `http://www.rfc-editor.org/rfc/rfc1321.txt`.

Theses.CZ. 2008–2009. Czech national archive of graduate theses. `http://theses.cz/`.

[5] As an example, the suspicious document 02010, characters 198,025 to 200,472 correspond to the source document 06059, characters 429,098 to 431,429 almost exactly. The development corpus can be downloaded from the competition web site.

[6] Refer to the competition web site for the exact description of the evaluation criteria, including the terms *recall* and *precision*.

[7] An example from the development corpus is the suspicious document 00002, characters 618,098 to 618,798, which are annotated to be similar to the source document 02400, characters 33,963 to 34,664.