

MatchDetectReveal: Finding Overlapping and Similar Digital Documents

Krisztián Monostori, Arkady Zaslavsky, Heinz Schmidt

School of Computer Science & Software Engineering

Monash University, 900 Dandenong Road, Caulfield East, VIC 3145, Australia

{*krisztian.monostori, arkady.zaslavsky, heinz.schmidt*}@infotech.monash.edu.au

ABSTRACT

The Internet provides easy access to large collections of semi-structured digital documents. WWW browsers, search engines and the "cut & paste" technique are tempting to substitute one's creativity by simple compilation from appropriate digital resources. This paper discusses the problems of detecting plagiarism in large collections of semi-structured electronic texts. Overlaps in and similarity of digital documents and software code are in the focus of this project. The conceptual architecture of the MatchDetectReveal system is presented along with possible applications. The main component of the system is using the string matching algorithms and a suffix tree representation. Both sequential and parallel cluster-based processing issues are addressed. The implementation and performance issues are also discussed.

KEYWORDS: Fast search algorithms, document overlap, information retrieval, plagiarism detection

1. INTRODUCTION

Digital libraries and semi-structured text collections provide vast amounts of digitised information on-line. Preventing these documents from unauthorised copying and redistribution is a hard and challenging task, which often results in avoiding putting valuable documents on-line (Garcia-Molina et al., 1995a). Copy-prevention mechanisms include distributing information on a separate disk, using special hardware or active documents (Garcia-Molina et al., 1995b). 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 engines students can easily find relevant articles and papers for their research. But this tool is a two-edge sword. These documents are available in electronic form, which makes plagiarism feasible by cut-and-paste or drag-and-drop operations while tools to detect plagiarism are almost non-existent.

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 (e.g. Plagiarism.org, 1999) and it is also possible that a student may potentially submit a plagiarised thesis. Garcia-Molina et al. (1996b) reports on a mysterious Mr.X who has submitted papers to different conferences, which 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.

There are several systems built for plagiarism detection including SCAM (Garcia-Molina et al., 1995b), Glatt (Glatt, 1999), plagiarism.org (Plagiarism.org, 1999). SCAM and plagiarism.org are similar in their approach. They build an index on a collection of registered documents by using hashing algorithms and comparing these hashed values. They report plagiarism if overlap is above a certain percent. On the contrary, Glatt does not use computationally intensive analysis but rather assumes that everyone has a different style of writing and students can more easily remember their own style than plagiarised sentences. Students are presented with their papers but every fifth word is eliminated and students are asked to fill in the blanks. According to Glatt (1999) no student has been falsely accused. This approach has the advantage that it can detect plagiarism even if the original papers are not present but has the drawback that students have to be involved. If we consider plagiarism detection for submitted conference papers, this approach is prohibitive.

Our approach uses computational analysis similarly to Garcia-Molina et al. (1995b) and Plagiarism.org (1999), but considers exact string matching algorithms rather than hashing, which has a finite probability of failure and is reported in Garcia-Molina et al. (1996a).

Obviously, string matching algorithms can only work on a limited number of documents, when used for text matching. This brings up another problem, namely the problem of finding candidate-documents in distributed digital libraries. This problem is also addressed in the dScam prototype developed at Stanford University and described in Garcia-Molina et al. (1996b).

Plagiarisers may also slightly modify the copied documents and this brings up another problem - detecting similar documents and measuring similarity. For example, Kock (1999) mentions a case of plagiarism where the only modifications made in the text included different numerical values and event whereabouts. String matching would not detect the overlap though the documents are very much similar. Similarity detection is therefore more complex and requires rule-based processing. In our project we are interested not only in finding overlapping and/or similar textual documents but also source code.

In this paper we discuss a copy-detection mechanism based on string matching and point out other applications of this mechanism. String matching algorithms have been widely used since early 70-s. String matching algorithms can be used in information technology applications, i.e. data compression, coding theory, file comparison. In this paper we will discuss the significance of string matching algorithms in digital libraries including copy-detection and plagiarism-detection in the context of the MatchDetectReveal (MDR) system.

This paper is structured as follows. Section 2 outlines the conceptual architecture of the MDR system, which is currently under development. Section 3 describes the matching engine component of the system and introduces suffix trees and their different applications including the problem of identifying the overlapping texts. It also discusses space and time requirements of building suffix trees. In Section 4 we analyse the efficiency of the matching engine in 3 hypothetical scenarios. We also discuss the algorithm of finding the overall plagiarised percentage of the suspicious document. In Section 5 we describe what we have reached so far and what is left for future work.

2. MATCHDETECTREVEAL SYSTEM ARCHITECTURE

In this section we outline the conceptual architecture of the proposed MatchDetectReveal (MDR) system. It is depicted in figure 1. Different issues of the matching engine are discussed in the following sections.

Users of the MDR system must be able to submit suspicious documents. MDR users will use an Internet-based interface for submitting documents. Submitted documents can be compared to the documents of the local repository or to documents on the Internet or other digital libraries e.g. ACM DL or IEEE DL. Once these documents are identified we can download them and compare to the suspicious document locally. Different indexing technics will be analysed to efficiently find candidate documents. We will also consider 'intelligent' ways of identifying candidate documents by e.g. checking the publications of authors listed in the references.

The matching engine is responsible for comparing the suspicious document to candidate documents. Section 3 discusses the matching engine. Here we note that the current implementation of the matching engine deals with exact matching and it will be supplemented with a 'Similarity and Overlap Rule Interpreter', which will define how to handle rephrasing, changing the names of localities, substituting synonyms for certain words.

With the constant increase of the processing power of commodity workstations clusters of workstations are widely used for parallel processing (Baker et al., 1999). Instead of sequentially analysing suspicious documents we can schedule these tasks on a cluster, which enables us to analyse more documents in parallel resulting in lower response time (Monostori et al., 1999).

Not only can we use special workstation clusters for processing documents in parallel but we can also utilise idle workstations. When we define the set of candidate documents we can also identify idle workstations in the close proximity of each document. Those workstations are likely to require less time to download documents they are responsible for and response time can again be reduced while workstation utilisation increases.

The visualizer component of the system is responsible for presenting the end user with the parts of the document, which are plagiarised. Currently overlap is defined as positions in the text and the length of the chunks, which will be converted into a visual representation in the future.

Document Generator is a supplementary component of the system, which generates sufficient number of documents to test our algorithm. You can define the size and overall plagiarism content of the document to be generated as well as the number of files to be used for plagiarism and the size of the chunks. It is also possible to define the number of words to be substituted by their synonyms to test more sophisticated ways of plagiarism.

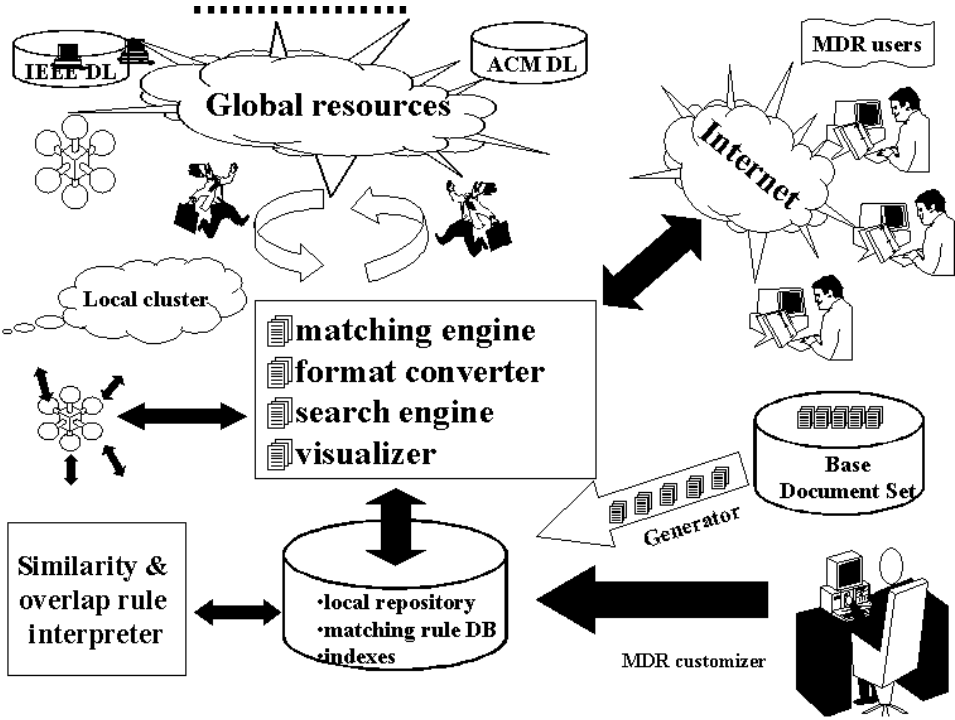


Fig.1. MDR architecture

3. THE MATCHING ENGINE

The matching engine of the MDR system uses suffix trees for comparing the documents. It is supplemented with a document converter and in the future it will also use the ‘Similarity and Overlap Rule Interpreter’ to handle similarities rather than exact matching.

The converter component converts different types of document into a unified format. Not only does it convert common file types (e.g. PS, PDF, MS Word, HTML) into plain ASCII but also plain ASCII formats must be further converted otherwise the system can easily be cheated e.g. by using different punctuations or adding extra newlines or whitespaces. The converter leaves all alphanumerical characters as they are, converting each alphabetical character to lowercase, and replace all other characters by another single character that is not alphanumerical. We convert any contiguous repetitions of these characters into a single character. We also shift numerical characters because we need a contiguous range of characters, so that the algorithm can work more efficiently. We also need a termination symbol for building the suffix tree, which can be chosen at either end of the range. Finally, we end up with a 38-character alphabet, which is important because the size of the suffix tree and the time of Ukkonen’s algorithm (Ukkonen, 1994; Gusfield, 1997) for building the suffix tree depend on the size of the alphabet. As an example let us convert the following text into our unified format:

Mr. X. plagiarised
a lot of documents
according to (Garcia Molina et al., 1996b)

After the conversion this text is:

mr`x`plagiarised`a`lot`of`documents`according`to`garcia`molina`et`al`W__\b`

The conversion algorithm ensures that chunks identical before conversion remain identical after conversion. Identifying the plagiarism content is based on the number of matching characters in converted formats.

Chang et al. (1994) describes an algorithm for finding so called *matching statistics* - $ms(i)$, which is the longest substring of T starting at position i that matches a substring somewhere in P. Having built a suffix tree for P, matching statistics of T can be found in $O(n)$ time where n is the length of T. The main idea of the algorithm is that starting from the first position in T we calculate the matching statistics $ms(0)$ by traversing the suffix tree of P. Then, in order to calculate $ms(i)$ while having calculated $ms(i-1)$ we have to back up to the node above our current position, follow the suffix link of that node and then traverse down from that node.

We have tailored both Ukkonen's building algorithm and Chang's matching statistics algorithm by taking into account that we only want to find overlaps starting at beginning of words. With this modification the space requirement can be reduced to approximately 20% of the original suffix tree and the matching statistics algorithm also benefits from this modification. This modification is similar to the one in Baeza-Yates et al. (1992) but they only suggest that substrings starting at the beginning of words are to be included and they do not consider how to build the tree in linear time. We analysed and modified Ukkonen's and Chang's algorithm to reflect these changes. We will also analyse the possibility of inserting only those suffixes that start at certain positions, e.g. every 30th, for further reducing the size of the suffix tree.

Part of a modified suffix tree for the converted string 'they`were`the`last`to`arrive`but`they`were`not`late`' is depicted in figure 2.

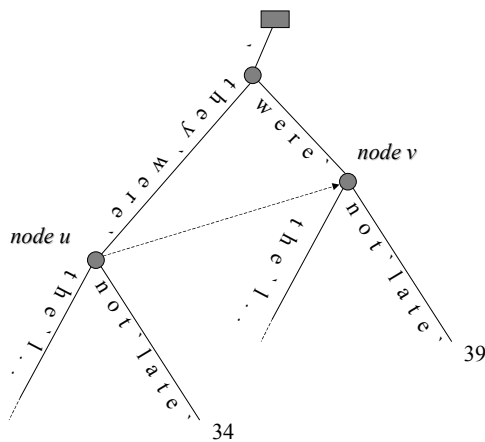


Fig 2. Part of a Modified Suffix Tree

If during the matching statistics algorithm we traverse down the route labelled by 'they`were`not`' in the next step we do not have to start matching at the root rather we follow the suffix link between *node u* and *node v* (depicted by the dashed arrow) and continue traversing the tree from *node v*. This interpretation of the suffix link is different from the one in the original Ukkonen's algorithm and since we only insert suffixes starting at the beginning of words a suffix link points to the node with the label of the original node without the first word. In our example the label of *node u* is 'they`were`' and the label of *node v* is 'were`'. Not only does this modified suffix tree require less space but the matching statistics algorithm can also benefit from it calculating the matching values only for positions where words start.

4. PERFORMANCE ANALYSIS OF THE SEQUENTIAL ALGORITHM

Two different approaches are described in (Monostori et al., 1999) to use suffix trees for identifying overlaps. If the matching statistics are found for each candidate document we have to calculate the overall number of plagiarised characters in the suspicious document to identify the overlap content of the document. In this calculation we ignore those matching chunks that are under a certain threshold. Our experiments showed that 60 characters is a reasonable value since names of organisations and institutes can easily exceed lower values.

We cannot simply add up all the values identified by the matching statistics algorithm because if there is a matching chunk of m characters at position i with the first word of length n then at position $i+n$ the algorithm will find a matching chunk of, at least, length $m-n$. We define an array with the same length as the length of the suspicious document. In this array we store the matching statistics value for each position. Next is the description of the algorithm finding the overall overlap of a document.

Our algorithm runs through the array and records the first non-zero value in a variable *last_length*. The starting position of this chunk is also recorded in *last_position*. The next nonzero value *current_value* at position *current_position* is compared to *last_length*. If this value defines a chunk that exceeds the range defined by *last_length*, that is *current_value* is greater than *last_value-(current_position-last_position)*, then this previous range bounded by *last_position* and *current_position* is added to an overall counter *overlap* while *current_value* and *current_position* is recorded as *last_value* and *last_position* respectively. The other possible case is that we cannot find a value exceeding the range defined by *last_position* and *last_value* in this case *last_value* is added to *overlap*. We also give a formal pseudo code of our algorithm. *msi* is the array of overall matching statistics.

```
last_position, last_value <- -1, 0
for i:=0 to 'length of suspicious document' - 1 do
  if msi[i]==0
    continue
  end if
  current_value, current_position <- msi[i], i
  if (current_position-last_position)>last_value
    overlap := overlap + last_value
    last_value, last_position <- current_value, current_position
  else if current_value>last_value-(current_position-
last_position)
    overlap := overlap + (current_position-last_position)
    last_value, last_position <- current_value, current_position
  end if
end for
overlap := overlap + last_value
```

During the process of calculating matching statistics we can record the name of that candidate document where the matching was found and also the position within the document can also be recorded. Practically we create a file for each candidate document where overlap is found and record positions in these files. Having analysed all candidate documents we can calculate the overall percentage of the document that is a copy of other documents and all the positions and documents are recorded. If this percentage is above a given threshold we will report the suspicious document as having considerable overlap with other documents.

There is one considerable drawback of this algorithm, namely, we have to build a suffix tree for each and every document. Building a suffix tree for a 1Mbyte document took about 1.1s on our test machine (Intel Pentium II 433MHz, 128M RAM, Windows NT Workstation). Monostori et al. (1999) describes an algorithm, which builds the suffix tree only once and compares candidate documents to that single suffix tree.

Below, the 3 common scenarios are described along with performance details of our algorithm.

Scenario 1: the suspicious document is genuine. There can be accidental overlaps but they do not add up above a given percentage.

Scenario 2: the suspicious document is heavily plagiarised, that is 60-70% percent of the document is copied from 8 different documents and the order of the chunks is mixed up.

Scenario 3: the most part of the suspicious document is genuine, but there is a huge chunk (1 page), which is copied from another document.

We set the chunk plagiarism threshold to 60 characters, the overall plagiarism threshold to 10%, that is if more than 10% of the document is copied from other documents then the suspicious document is reported as plagiarised. We also set a threshold for a large plagiarised chunk to 1000 characters meaning that regardless of the overall percentage finding a chunk of this size will report the document as plagiarised.

The files in scenario 1 and 2 are 11K and 14K respectively, while the file in scenario 3 is a 1.67M document. We compare these three documents to 19 documents, which are 3.5M altogether and of course contain those 8 documents, which are used in Scenario 2 and the one used in Scenario 3. We used the algorithm, which builds only one suffix tree, for analysing the documents. This algorithm correctly reported the files in Scenario 2 and 3 as plagiarised while the document in Scenario 1 as genuine.

The running time for calculating matching statistics in Scenario 1 and 2 were almost the same (see table 1). But Scenario 3 took more time. It contradicts with the linear time bound of the matching statistics algorithm, which states that the running time of the algorithm is independent from the size of the tree and only depends on the strings compared to the tree, which are the candidate-documents of 3.5M in our case. We found two issues, which influence the running time of the algorithm in practice. Firstly, the suffix tree in Scenario 1 and 2 can fit into the cache, which considerably speeds up the algorithm. Secondly, it is true that the so called skip/count steps (Gusfield, 1997) are bounded by 3m but the bigger the tree the more the skip/count steps, which adds up to the running time. It is still true that the algorithm has linear time worst-case bound but in reality the case is worse if the tree is bigger.

Scenario	Size	$i/o_{\text{suspicious}}$	t_{suffix}	i/o_{cand}	Msi	msi_{overall}	$t_{\text{positions}}$
1	309,192	60	10	1733	530	7	-
2	228,716	60	10	1642	531	6	120
3	41,573,992	1191	2153	1922	1923	-	691

Legend:

Size: Size of suffix tree (bytes)

$i/o_{\text{suspicious}}$: i/o time to read suspicious document (ms)

t_{suffix} : Time to build suffix tree (ms)

i/o_{cand} : i/o time for reading candidate documents (ms)

msi: Calculating msi by document (ms)

msi_{overall} : Calculating overall msi (ms)

$t_{\text{positions}}$: Calculating positions (algorithm 1 – without i/o time) (ms)

Table 1. Performance analysis details for 3 scenarios

5. CONCLUSION AND FUTURE WORK

In this paper we described the proposed conceptual architecture of the MatchDetectReveal (MDR) system. The core component of the system (matching engine) was also described. We gave a performance analysis of the algorithm implemented in Visual C++ using suffix trees and Ukkonen's as well as Chang's algorithms. These two algorithms were modified to use a more space-efficient suffix tree representation of the text.

The matching engine will be modified to be able to detect more sophisticated ways of plagiarism i.e. substituting words with synonyms changing the names of localities, changing certain numbers. This will be implemented by using a set of rules, which will be interpreted by the 'Similarity and Rule Interpreter'. The 'Document Generator' component will also be modified to create documents using the rules mentioned above. Currently 'Document Generator' is capable of copying exact chunks into a random text with possibly substituting words by their synonyms.

Section 4 describes the space requirement of a suffix tree, which can be prohibitive in the case of very large documents. Efficient parallel algorithms must be analysed to build and use suffix trees for the matching statistics algorithm. This algorithm has to be able to distribute the suffix tree structure among different nodes of the cluster. Another reduction of the size of a suffix tree is to convert it into a directed acyclic graph (DAG) but we have to analyse how the matching statistics algorithm can be applied on a DAG.

As today information is highly distributed and we have access to distributed resources we will analyse the mobile agent technologies to use our algorithm. One possible application is to send out an agent to different sites with the suspicious document, build the suffix tree on that site and compare candidate-documents there. After having analysed that site, the agent can return the information on overlapping for that site and the overall statistics can be calculated at a central site.

REFERENCES

- Baeza-Yates R. A., Gonnet G. H., Snider T. (1992). New Indices for Text: PAT Trees and PAT Arrays, in Frakes W. B., Baeza-Yates R. A. *Information Retrieval: Data Structures & Algorithms*. (Prentice Hall) pp. 66-82.
- Baker M., Buyya R. (1999). Cluster Computing at a Glance in Buyya R. High Performance Cluster Computing. (Prentice Hall) pp. 3-47.
- Chang W.I., Lawler E.L. (1994). Sublinear Approximate String Matching and Biological Applications. *Algorithmica* 12. pp. 327-344.
- Garcia-Molina H., Shivakumar N. (1995a). The SCAM Approach To Copy Detection in Digital Libraries. *D-lib Magazine*, November.
- Garcia-Molina H., Shivakumar N. (1995b). SCAM: A Copy Detection Mechanism for Digital Documents. *Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95), June 11 - 13, Austin, Texas*.
- Garcia-Molina H., Shivakumar N. (1996a). Building a Scalable and Accurate Copy Detection Mechanism. *Proceedings of 1st ACM International Conference on Digital Libraries (DL'96) March, Bethesda Maryland*.
- Garcia-Molina H., Gravano L., Shivakumar N. (1996b). dSCAM: Finding Document Copies Across Multiple Databases. *Proceedings of 4th International Conference on Parallel and Distributed Information Systems (PDIS'96), Miami Beach, Florida*.
- Glatt Plagiarism Screening Program (1999). URL <http://www.plagiarism.com/screen.id.htm>.
- Gusfield D. (1997). *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*. (Cambridge University Press)
- Kock, N. (1999). A case of academic plagiarism. *Communications of the ACM*, July 1999, Vol 42, No.7, p.96-104.

Monostori K., Schmidt H., Zaslavsky A. 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.*

Plagiarism.org, the Internet plagiarism detection service for authors & education (1999). URL <http://www.plagiarism.org>

Ukkonen E. (1995). On-Line Construction of Suffix Trees. *Algorithmica 14*. pp 249-260.