Using the MatchDetectReveal System for Comparative Analysis of Texts

Krisztián Monostori, Arkady Zaslavsky

School of Computer Science and Software Engineering
Monash University, Melbourne, Australia
900 Dandenong Rd, Caulfiled East
VIC 3145, AUSTRLIA

{Krisztian.Monostori, Arkady.Zaslavsky}@csse.monash.edu.au

Alejandro Bia

Miguel de Cervantes Digital Library

University of Alicante, Alicante, Spain
Ap. Correus 99
E-03080 Alacant, SPAIN

abia@dlsi.ua.es

Abstract

In this paper we are introducing the MatchDetectReveal system, which is capable of identifying the similarity between documents. Different applications of the system are discussed including cross-referencing multiple editions of literary works, plagiarism detection, organizing collections of documents and comparative analysis of texts. The system uses suffix trees and suffix vectors for comparing documents. These data structures are very fast and powerful, which allows fast comparison of documents. The front-end of the system is fully Web-based, thus users only need to use a Web browser to access the system. The results are also presented as HTML files utilising the hyperlink capabilities of HTML documents.

Keywords document comparison, plagiarism detection, digital libraries.

1 Introduction

Digital libraries provide vast amount of digitised information on-line. These documents may appear on multiple computers all over the Internet for different purposes. These include illegal copies of documents and plagiarised documents. Finding these documents and presenting the overlap between these documents are challenging tasks. The MatchDetectReveal (MDR) system uses a search-engine to identify a set of candidate documents. Once this set is identified the matching engine compares the documents and presents the results to the user.

There are other applications of the matching engine, which we have discovered in a joint project with the Miguel de Cervantes Digital Library [1]. The Miguel de Cervantes DL is the biggest collection of digital literary texts in Spanish. The project aims to join the skills and experiences of humanists and computer scientists to develop tools and methods for digital libraries. We have identified four areas where MDR could be applied in the Miguel de Cervantes DL, which are discussed in the following subsections.

1.1 Detecting cross-references

The first application we thought of was to automatically detect quotations or cross-references between different texts. In this case the quote could be hyper-linked to the original. We can imagine many research situations where automatic detection of similar sections of text can be useful. It could be a valuable aid in history research for instance.

1.2 Organizing collections of small pieces of text

Another use is to detect repeated poems in different poem collections (may also be tales, letters, etc.), where some of these textual units are repeated in different editions. Most often, when collections of poems are developed, it is difficult to keep track of which poems have been included and where, and which not. MDR proved to be a helpful tool for detecting and locating the corresponding pieces of text for verification purposes.

1.3 Comparative analysis of texts

The objective is to compare different editions of the same literary work. Philologists are usually interested in this kind of comparisons for research purposes. In ancient literature, there usually are different editions of the same work, with modifications performed some times by the author, some others by the editor. This comparative analysis is in itself an interesting but tedious field of study where any kind of automation is

Proceedings of the 6th Australasian Document Computing Symposium,
Coffs Harbour, Australia,
December 7, 2001.

welcome. To cite an example, there are various ancient editions of the *Quijote de la Mancha*, all originals from Cervantes, but still different. This application differs from the previous ones, in the sense that here we have to find differences rather than matches. Although in this case the matching strings are usually in the same order in both sources, synchronization is not always possible since the sources may have long additions inserted in different places, apart from small differences in the matching zones. A conventional sequential comparison may also fail in this case.

1.4 Detecting variations and mistypings

There are variations in spelling owed to ancient editor changes, which are not strictly errors since the Spanish language was not normalized until 1713, where the Real Academia de la Lengua Española was created and language spelling started to settle. Those variations, though legal, produce differences between editions.

On the other hand, no matter how much care correctors put on freeing texts from digitisation errors, there are always some errors that remain. So an unforeseen result of the application of MDR to comparative analysis was the detection of spelling errors or variations in DL texts, out of the comparison of different editions of the same work.

2 The MatchDetectReveal system

In this section we introduce the MatchDetectReveal (MDR) system [8] and the following two sections contain detailed discussions of the two most important components: the matching engine and the visualiser. The architecture of the MDR system is shown in figure 1.

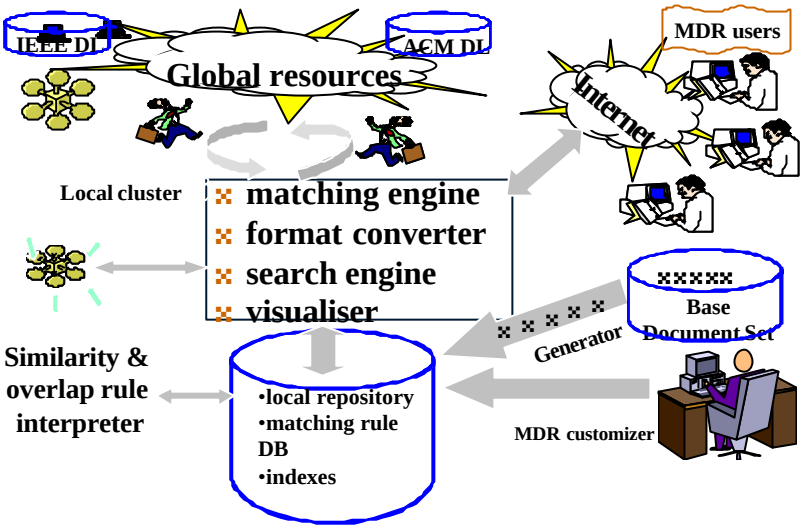

Figure 1. MDR architecture

Documents must undergo pre-processing before they are compared. In the pre-processing phase documents are converted to plain ASCII format and then they are further converted to a format suitable for the search engine. More about this format can be found in the following section. It is also the task of this component to eliminate the slight alterations of texts: every character that is not a letter or a number is converted to a single character, thus we eliminate such alterations as using different punctuations, adding extra spaces, or adding extra new line characters.

Once a document is pre-processed it is chunked using a similar technique to that of the SCAM system [5]. It is the task of the search-engine to identify candidate documents based on the chunks extracted from each document.

Once we have a set of candidate documents they can be processed by the matching engine. The matching engine uses suffix trees and suffix vectors to identify overlapping chunks between documents. For a detailed discussion of the matching engine see the following section.

The output of the matching engine component is a list of positions and lengths of chunks that are found identical. The visualiser component takes these positions and length values as its input and produces a HTML- and JavaScript-based output. Different possible presentations of the results using the visualiser component are discussed in Section 4.

The document generator is a complementary component that can be used to generate documents that overlap with documents in the base document set. These documents then can be used to test our algorithms. Here we note that our system has been tested by using "real" documents as well as documents generated by the document generator component.

Using the local cluster for the comparison is discussed in [9]. Comparison jobs can be generated when a suspicious document is submitted and the local cluster can be used to compare documents in

parallel. Also global grids may be used to compare documents utilising resources in the close proximity of documents.

3 Matching engine

The matching engine component compares documents that have been pre-processed by the converter component. It uses suffix trees and suffix vectors for comparison, which are briefly discussed here.

A suffix tree is a data structure that includes every suffix of a given string, thus every substring. Suffix trees can be built in linear time using different algorithms [6][11]. We used Ukkonen's algorithm not only because of its relative simplicity but because it also stores suffix link information, which is heavily utilised by the matching statistics algorithm. The suffix tree of the string S='abcdabca$' is shown in figure 2. The '$' sign is the unique termination symbol that is required by the algorithm.

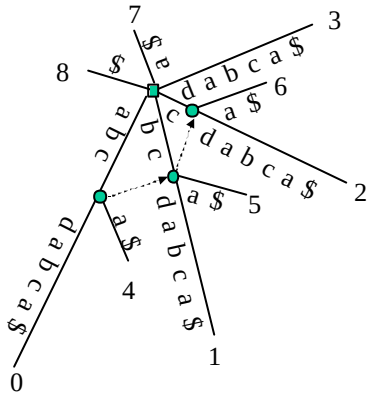

Figure 2. Suffix tree example

Chang et al. [3] 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.

Without going into details of the matching statistics algorithm we show how the first two steps are performed on our example string. Let us suppose that we have built the suffix tree of figure 2 and we want to find the overlap of that string with T='abcda$'. ms(0) is the length of the longest chunk starting at position 0 in T that matches a chunk anywhere in S. We find this value by traversing down the tree starting from the root (depicted by the square). We find that 5 characters ('abcda') match, that is we are at letter 'd' on leaf 0. We back up to the closest node and follow its suffix link (depicted by the dashed arrow), which places us on the route of 'bc'. From here we can only match 'da', so we set the matching statistics value ms(1) to 4. The rest of the steps can be performed similarly. With some extra information on the nodes we can even locate the start position of the overlap in the original document. Having identified the start positions and the matching statistics values we know the overlapping chunks.

Suffix trees are very space-consuming data structures [9] but the space requirement can be reduced by applying different techniques. The first technique that we may apply stems from the structure of natural language texts. We are not interested in overlapping chunks that start in the middle of words, thus we only insert suffixes of the text, that start at the beginning of a word, into the suffix tree [9]. An example is shown in figure 3 ('+' symbols represent spaces).

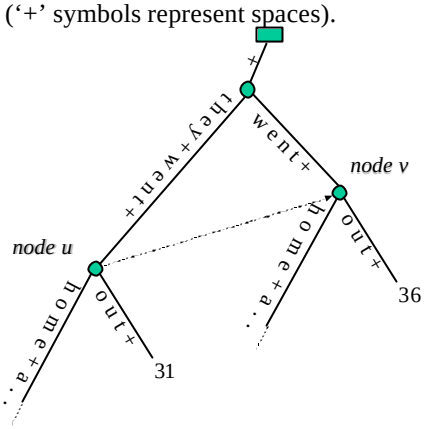

Figure 3. Example of reduced suffix tree

Another way to reduce the space requirement is to use suffix vectors [10]. Suffix vectors are alternative structures for suffix trees and they store edge information aligned with the string itself, which allow more space-efficient storage and it also eliminates some redundant information stored in the tree, thus speeding up the matching statistics algorithm. The suffix vector of string S='abcdabca$' is shown in figure 4.

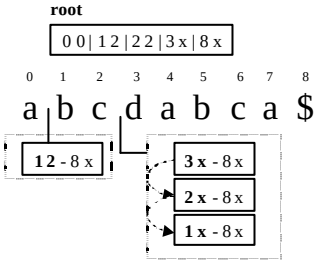

Figure 4. Example of a suffix vector

4 Visualizer

The visualiser component takes the output of the matching engine and generates HTML files that show the overlap between documents. We have identified

two different interfaces, which are currently being tested by librarians in order to evaluate the effectiveness of them.

The first interface uses different colours to denote the overlapping chunks. Ten different colours are identified and they are reused if more than 10 overlapping chunks are present in the document. Each chunk is also a link and by clicking on the link we jump to that section of the candidate document where the matching chunk can be found. Figure 5 shows an example output.

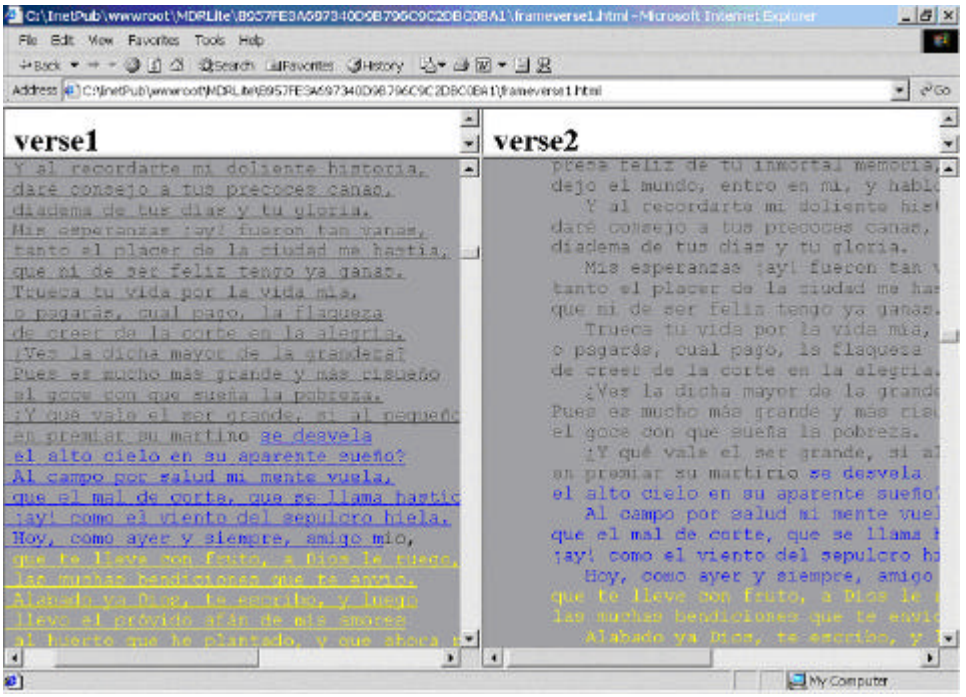

Figure 5. Output of the visualiser component

The system can compare one document to many documents and present the overlap in a window with four panes. The top panes contain the name of the documents shown in the bottom panes. The left panes show the original document while the right panes the current candidate document. The current candidate document may change because the original document may overlap with more than one document and the right panes always contain the corresponding candidate document and the name of that document.

The visualiser takes the position and length values, which are identified for the converted document, and calculates the positions in the original document, which may be different since multiple whitespaces are eliminated in the converted document. Then it places HTML tags in the document to mark up overlapping chunks. It also places the relevant colouring tags.

In the other interface the basic look-and-feel is the same as the previous one, though we make use of the features of JavaScript. In this interface each overlapping chunk is depicted by the same colour and once the mouse is over a given chunk it changes its colour, a tooltip shows information on that chunk and in the right pane the candidate document is automatically scrolled to the overlapping chunk and its colour also changes. The information shown in the tooltip includes the percentage of text the given chunk represents, the total number of overlapping chunks in the document, the sequence number of the given chunk, the position in both the original and the candidate document, the length of the chunk in both documents, and the overall size of both documents.

The process of generating these documents are very similar to the previous method the only difference is that the markups that need to be inserted are more complex because they contain JavaScript code beside standard HTML tags. An example of this second interface is shown in figure 6.

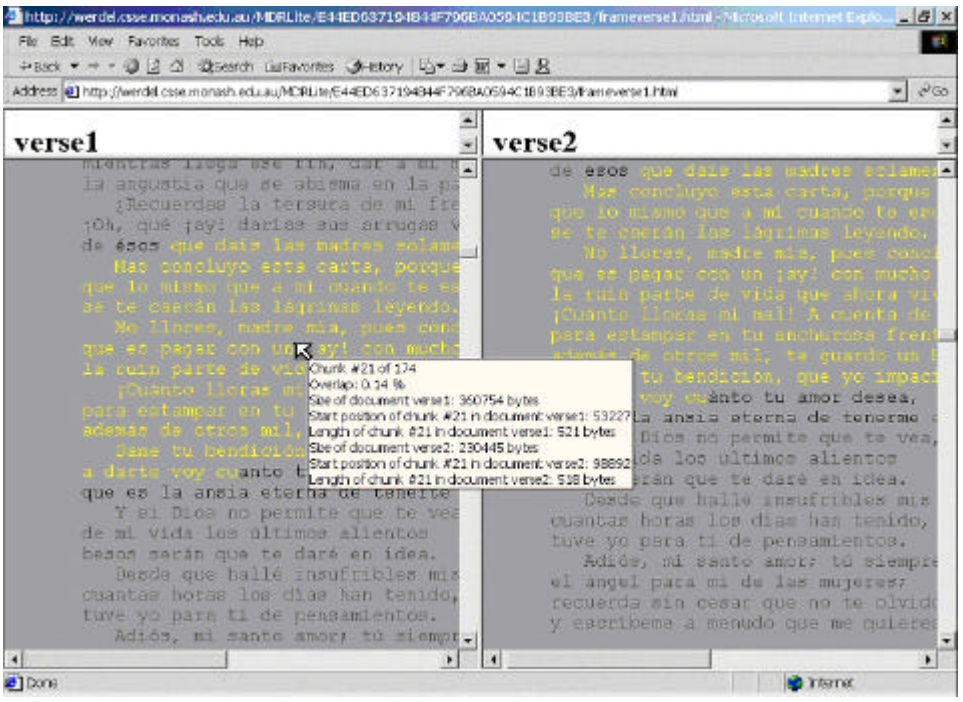

Figure 6. Output of the visualiser

5 Document Submission

A comprehensive document submission component is currently under development. This submission component will allow not only the submission of isolated documents but it will also be adapted to existing document submission systems used for e.g. assignment submission.

The current interface uses a file upload page. In that page you can select a file from your hard disk that is uploaded to the server. The file must be a zip file containing compressed plain ASCII text files. Non-ASCII files are discarded by the system. Once the file is uploaded the MDR system running on the server decompresses the zip file and does the necessary conversion that has been discussed in Section 3. The text files are then compared pair-wise using the matching engine. The output of the matching engine is fed to the visualiser, which generates the HTML files. These HTML files are stored for a week on the server and removed after that period has expired. Users are given the option to download all HTML files compressed into one zip file. Then that file can be decompressed on the users machine and the user can check the results on his/her own computer without having to be connected to the network. The document submission interface is shown in figure 7.

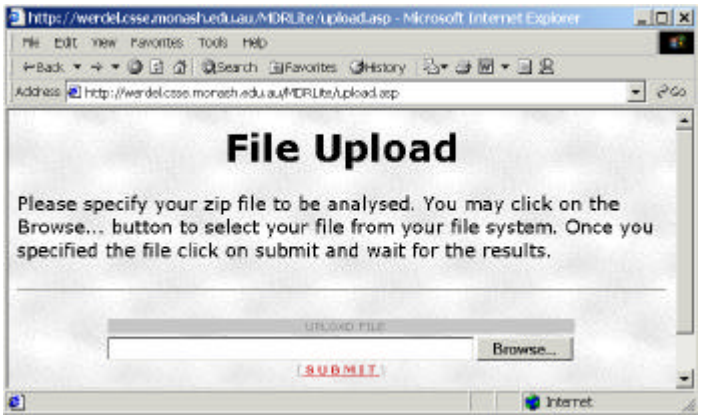

Figure 7. Document Submission

There are a few parameters that can be set for the system. These parameters are fixed at the time of writing but later on we will implement an interface where they can be set. The first of these parameters is the data structure to be used. We have three choices: suffix tree, reduced suffix tree, suffix vector. Different applications may require different data structures. For instance, when DNA sequences need to be compared

we cannot use the reduced suffix tree structure because DNA sequences have no such natural boundaries as word boundaries in natural language text. Another adjustable parameter is the character set to be used. The current setting uses the English alphabet but as our examples from the Miguel de Cervantes DL show character sets of other languages can be adapted. You can also set the minimum number of characters that you consider an overlap. Of course, if it is set to 1 you have a fairly good chance that the original document will show 100% overlap because each single character appears in one of the candidate documents. The default value of this parameter is 60 characters. This value is close to the value used in other systems with similar purposes [2] [4] [7].

6 Results of comparisons in the Miguel de Cervantes DL

In Section 1 we have discussed different applications of the MDR system in the Miguel de Cervantes DL. In this section we give a few examples to demonstrate the applicability of the system.

One of our first experiments was to compare a couple of collections of poems from Ramón de Campoamor y Campoosorio. We could easily keep track of the poems that appeared in both collections. Figure 8 shows the cross-reference of one isolated poem that appears in two different poem selections.

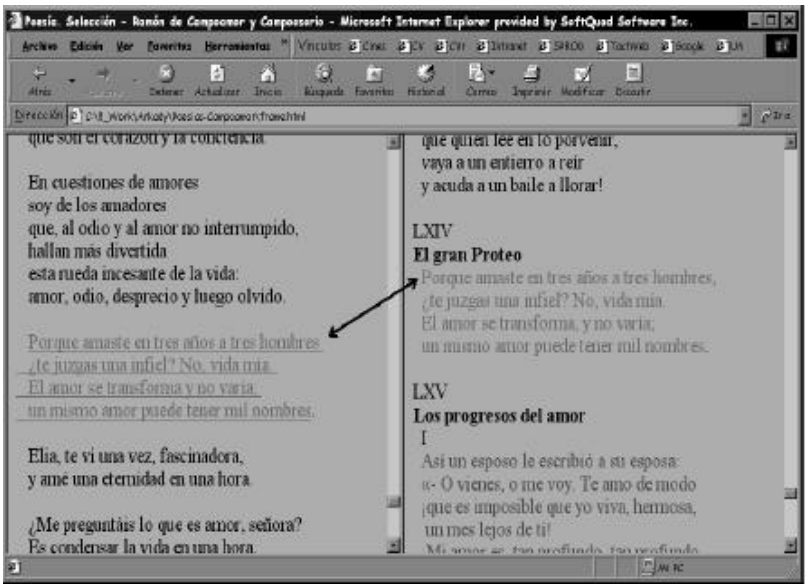

Figure 8. Comparison of different poem collections

As an example of the problem discussed in Section 1.3, there are various ancient editions of the Quijote de la Mancha, all originals from Cervantes, but still different. In the example of figure 9, we can see clear differences in two versions of Don Quijote: on the left frame we can read "*Eran cuatro, y venían con sus quitasoles, con otros cuatro criados a caballo y dos mozos de mulas a pie*", while on the right one we see "*Eran seis, y venían con sus quitasoles, con otros cuatro criados a caballo y tres mozos de mulas a pie*". The merchants from Toledo, in one case are six, in the other four, and their servants on foot are two on one side and three on the other. A little bit above there is another difference in meaning. On the left it says "muy pensado" (thoughtfully) and on the right "muy bien pensado" (very well thought). These are examples of changes introduced by editors through the centuries.

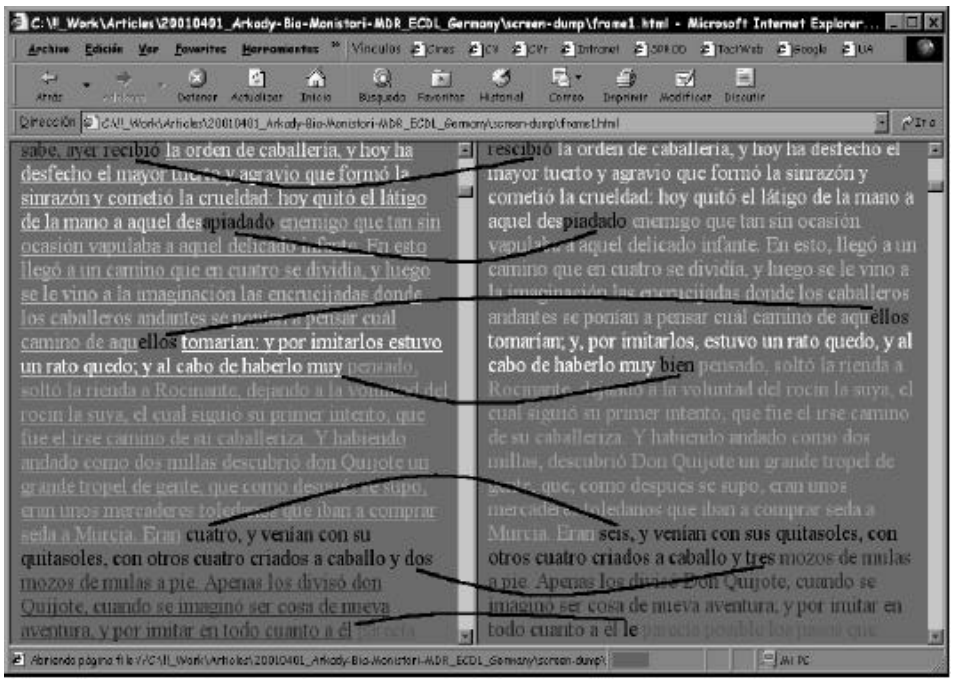

Figure 9. Comparing two editions of Don Quijote

7 Conclusion and future work

In this paper we have presented the MatchDetectReveal (MDR) system, which has originally been developed for plagiarism detection. This paper has discussed digital library-related implications. We have shown that four different areas can be successfully addressed by the system: detecting cross references, organizing collections of small pieces of text, comparative analysis of texts, and detecting variations and mistypings.

The advantage of the MDR system over existing tools that it does not rely on sequential comparison, which means that chunks appearing in different order in two different texts are identified by the system.

We have also described the architecture of the MDR system and the two most vital components have been discussed in detail. We presented two different possible outputs of the system, both of them are HTML-based with one using JavaScript.

We are currently working on a visualiser component that would generate an XML-based output. In the output document the overlapping chunks would be marked up by using the XML syntax, and transformation tools could be used to generate any kind of required output including HTML.

Acknowledgement

Support from Distributed Systems Technology Centre (DSTC Pty Ltd) for this project is thankfully acknowledged.

References

[1] A. Bia and A. Pedreño. The Miguel de Cervantes Digital Library: The Hispanic Voice on the WEB. *LLC (Literary and Linguistic Computing) journal*, Oxford University Press, 16(2): 161-177, 2001.

[2] A. Z. Broder, S. C. Glassman, and M. S. Manasse. Syntatic Clustering of the Web. *Sixth International Web Conference*, Santa Clara, California USA. URL http://decweb.ethz.ch/WWW6/Technical/Paper205/paper205.html

[3] W. I. Chang and E. L. Lawler. Sublinear Approximate String Matching and Biological Applications. *Algorithmica 12*. pp. 327-344, 1994.

[4] H. Garcia-Molina, N. Shivakumar. 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.

[5] H. Garcia-Molina, N. Shivakumar. The SCAM Approach To Copy Detection in Digital Libraries. *D-lib Magazine*, November, 1995.

[6] D. Gusfield. Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology. Cambridge University Press, 1997.

[7] N. Heintze. 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] K. Monostori, A. Zaslavsky, H. Schmidt. 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] K. Monostori, A. Zaslavsky, H. Schmidt. 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] K. Monostori, A. Zaslavsky, I Vajk. Suffix Vector: A Space-Efficient Suffix Tree Representation. *International Symposium on Algorithms and Computation*, Christchurch, New Zealand, Dec 19-21, 2001. Accepted for publication.

[11] E. Ukkonen. On-Line Construction of Suffix Trees. *Algorithmica 14*. pp 249-260, 1995.