Lasse Lehmann and Arno Mittelbach and Christoph Rensing and Ralf Steinmetz: *Automatic Detection and Visualisation of Overlap for Tracking of Information Flow*. In: Klaus Tochtermann and Hermann Maurer: Proceedings of I-KNOW 2010, 10th International Conference on Knowledge Management and Knowledge Technologies, p. 186-198, Verlag der Technischen Universität Graz, Austria, September 2010.

Automatic Detection and Visualisation of Overlap for Tracking of Information Flow

Lasse Lehmann[1], Arno Mittelbach[1], James Cummings[2], Christoph Rensing[1] and Ralf Steinmetz[1]

[1]KOM Multimedia Communications Lab
Technische Universität Darmstadt
Rundeturmstr. 10, 64283 Darmstadt, Germany
{lasse.lehmann, arno.mittelbach, christoph.rensing, ralf.steinmetz}@kom.tu-darmstadt.de

[2]Research Technologies Service
Oxford University Computing Services - University of Oxford
13 Banbury Road, Oxford, OX2 8NP, UK
James.Cummings@oucs.ox.ac.uk

Abstract: The detection of redundant or reused passages in texts is an important basis for various tasks including tracking of information flow, plagiarism detection, origin detection, web search and information retrieval. Being able to track the evolution of a piece of information through different revisions or instances of documents can generally help to gain an impression of the document's background. In this paper we propose an efficient algorithm for detection of textual overlap between documents as well as a tool for its visualisation, created in the course of the *Holinshed Project* at the University of Oxford. The Evaluation on an annotated corpus shows that the proposed algorithm performs better than state of the art approaches.

Keywords: Information Flow, Information Tracking, Overlap Detection, Overlap Visualisation, String Matching
Categories: H.3.3, H.3.1, H.5.2

1 Introduction and Motivation

The detection of redundant or overlapping passages in texts is an important basis for various tasks including tracking of information flow [Metzler, 2005] [Kim, 2009], plagiarism detection [Clough, 2003], origin detection, web search and information retrieval [Hamid, 2009]. Being able to track the evolution of a piece of information through different revisions or instances of documents can generally help to gain an impression of the document's background. When considering information flow in historic works like e.g. Holinshed's Chronicles or the Bible, it is even possible to draw conclusions about historic events or changes in attitude by studying how and when information has been changed or adapted.

In this paper we present an algorithm for the detection of overlap between textual documents as well as an intuitive tool that can be used for its visualisation and annotation. Both have been created during the *Holinshed Project*. One main goal of this project was to detect, localize, annotate and visualise the overlap between the two versions of Holinshed's famous Chronicles, which we describe in Section 2. Section 3 is dedicated to overlap detection. Herein we propose an approach for detection and

localization of textual overlap and compare it to existing state of the art techniques using a corpus created in the course of the *Holinshed Project*. In Section 4 we describe the TEI Comparator, a tool that can be used for the visualisation and annotation of overlapping text, while Section 5 concludes the paper and gives an outlook on possible future work.

2 The Holinshed Project

The *Holinshed Project* at the University of Oxford has created an online parallel text edition of Holinshed's Chronicles of England, Scotland, and Ireland. This sixteenth-century work was at once the crowning achievement of Tudor historiography and the most important single source for many contemporary playwrights and poets, above all Shakespeare, Spenser, Daniel, and Drayton. Although the work is popularly known as Holinshed's Chronicles, it was necessarily a collaborative endeavour, the authors and revisers contained a range of individuals.

The work was first printed in 1577 and then a much revised and expanded second edition was printed in 1587. The difference between these two editions is significant and the revisions help us understand the changing attitudes of the day. Although the significance of the importance of Holinshed's Chronicles in helping us to understand Elizabethan literature, politics, and history is undisputed, there has not been a complete scholarly edition of them or an in-depth systematic analysis. The aim of the *Holinshed Project* is to stimulate new interest in the Chronicles, through first publishing a secondary handbook on the making, transmission, reception, appropriation, and literary and historical significance of the Chronicles as well as eventually creating a complete original-spelling annotated edition [Holinshed, 2010].

In order to undertake any systematic investigation of the two editions of Holinshed's Chronicles there are many stumbling blocks. One of the most important is simply being able to trace the changes from one edition to the other. In the absence of an existing critical edition detailing the movement, reorganisation, fragmenting, amalgamation, excising and rephrasing of all aspects of the text it was decided that it would be helpful to have a very basic set of electronic texts to commence with in order to start work on the secondary handbook. These texts would then have to be linked or aligned in some manner to allow jumping from any paragraph-level object to the corresponding one(s) in the other edition often moved quite a distance from its original location. By doing this, authors of articles for the secondary volume about Holinshed's Chronicles could compare what had happened to certain portions of the document from one edition to the other concerning the particular topic they were writing about and thus draw conclusions about changes in attitude of the authors or the political landscape of that time.

Literature published about the Holinshed Project and Holinshed's Chronicles are available at http://www.cems.ox.ac.uk/holinshed/bibliography.shtml. The texts themselves have been released publicly and are available at: http://www.english.ox.ac.uk/holinshed/ and free for academic re-use. Also available is an HTML version that has been created on the basis of above mentioned analysis, which allows for navigation between matching portions of the two editions.

3 Overlap Detection

One of the first goals of the *Holinshed Project* was to create a parallel text edition that was to form the basis for an in-depth analysis of the two editions of Holinshed's Chronicles. This meant that overlapping text sequences between the 1577 and 1587 version had to be identified and localised. As the two editions each consist of almost 20,000 paragraphs (roughly 2.5 million words in the 1577 edition and almost 4 million words in the 1587 edition) it was decided to start with an automatically created initial comparison that could then be refined by domain experts. There are several existing approaches that can be used for the detection of textual overlap in the given scenario. Section 3.2 describes these state of the art algorithms. The algorithm we have designed for this task is described in section 3.3 while Section 3.4 covers the evaluation we did on the corpus created in the course of the *Holinshed Project*.

3.1 Definitions and Preprocessing

Before the overlapping sequences in two texts can be determined, the texts usually need to be preprocessed. Typical preprocessing steps include cleaning (removal of special characters and/or punctuation), case-folding, stemming and stopword filtering. In case of the Chronicles difficulties were for example a difference in spelling between the two editions (and sometimes even within one edition) or the substitution of similar looking letters for one another (e.g. 'v' instead of 'u' or a '~' instead of 'n'). We found that a drastic cleaning step worked best, where for example all vowels were removed and special characters were replaced by their canonical counterpart (exchanged all '~' with 'n').

Usually a text is then divided into *tokens*. When the input strings represent natural language texts, a token is typically a word, as for DNA sequences usually character based tokenization is used as each character in a DNA string represents a special nucleotide.

An *n-gram*, also called *shingle*, is a sequence of *n* consecutive tokens extracted from the text. Transforming the title - *The description and Chronicles of England* - of one of the books within the Chronicles into 3-grams would hence result in [The description and], [description and Chronicles], [and Chronicles of], [Chronicles of England]. A *match* between two strings A and B is a string which is substring of both A and B.

A commonly used symmetric measure of similarity is the Jaccard measure [Jaccard, 1901] which is defined as the intersection of A and B divided by the union of A and B:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

The problem herewith is that differences in length between the two strings A and B are not reflected in the measure. If, for example, a short string A is entirely included in some longer string B then the Jaccard measure will indicate that the two strings are not very similar.

Another approach to determine if two strings overlap is to calculate their containment [Broder, 1997]. Since containment is an asymmetric measure it can be calculated in two ways, depending on the length of which string is used as divisor, thus reflecting differences in length of the two input strings: The containment of string A in string B is calculated by dividing the intersection of A and B by the length of string A and vice versa:

$$C(A,B) = \frac{|A \cap B|}{|A|}$$

3.2 Related Approaches

N-gram-overlap [Lyon, 2001] is a basic approach for detecting overlapping passages in strings. In a first step overlapping n-grams (shingles) are extracted from the strings by sliding a window of size 'n' over both strings. The shingles are stored (e.g. in lists) and the sets of n-grams of both texts compared. To determine the containment of two strings the number of matching shingles, i.e. the intersection of the two shingle sets is divided by the number of shingles in string A or B respectively. N-gram-overlap does not take into account the order of shingles in both strings. To do so, an offset has to be stored in addition to each shingle. N-gram-overlap runs in linear time.

Greedy String Tiling (GST) proposed by Wise [1993] is a heuristic that approximates the maximal tiling of two strings. A *tile* is an association of a substring of string A with an equal substring of string B. Each token of a string can only be present in one tile. A *tiling* with minimal tile length *m* between strings A and B is a list of unique tiles where the length of each tile is *m* or greater. The GST algorithm searches for matches of maximal length, marks them as tiles and then continues the search ignoring the marked tiles until no more matches with the given minimal length can be found. The heuristic prefers tiles of maximal length over an optimal tiling of two strings, e.g. a tiling with one tile of length six is preferred over a tiling with two tiles of length three and four. If, for example, the minimal tile length is 3 and a tiling was to be generated for “ACCAAT” and “ACCACAAT” then GST would choose a single tile “ACCA” rather than the two smaller tiles “ACC” and “AAT”. Containment can now be calculated by dividing the overall length of all tiles in the tiling by the length of strings A or B respectively. The GST heuristic has a worst case complexity of $O(n^3)$ [Loose, 2008].

A classical problem in computer science is the identification of the **longest common subsequence (LCS)** [Maier, 1978] of two (or more) strings. A longest common subsequence is the longest sequence of common tokens of two input strings where the order of tokens has to be preserved but non matching tokens may be skipped. Usually tokens on character basis are used to determine the LCS. The LCS of two strings "ACCAAGCT" and "TACCCGCAT" thus is "ACCGCT". When the longest common subsequence is known, its length can be divided by the length of either of the strings to calculate containment measures. This approach suffers from the fact that it is extremely prone to changes in the order of the strings to compare. When the position of one sentence changes, the resulting subsequence can be significantly different. Thus we do not expect the approach to perform well in the given use case.

3.3 ShingleCloud Approach

ShingleCloud is an extended and adapted version of the n-gram-overlap approach. It is designed for efficient detection and localisation of overlapping sequences between two strings. The intended use case of the algorithm is a needle-haystack search, where it has to be determined whether, where, and to which extend a comparatively short string A (the needle) is present within a longer string B (the haystack). However it works equally well if needle and haystack have approximately the same length.

After preprocessing the input strings, shingles are extracted from both strings and stored in lists. For performance considerations the shingles extracted from the needle should be stored in a data structure optimized for fast lookups (e.g. a hash table). The shingle size (*n*) is thus the first of three relevant parameters used by the algorithm. In the next step the so called ShingleCloud, in its simplest form a sequence of bits, is created. This is done by traversing the shingles of the haystack in the order they appear and adding one bit to the ShingleCloud for each traversed shingle. A one is appended, if the n-gram is also present within the needle, i.e. is contained in the corresponding list, and a zero is appended otherwise.

Based on the resulting ShingleCloud, it can be determined manually or automatically whether and where parts of the needle are present within the haystack. A simple ShingleCloud could be, for instance:

000000000000011100000000111100000000000

In this example the needle has been split up into two parts. If we assume that 3-grams were used (i.e. the shingle size is 3) then we could, for example, conclude that the first part of the needle can be found at position 15 (13 consecutive zeros represent 15 tokens if *n=3*). If we knew more about the needle, such as for example its length, we could further conclude whether it is completely enclosed in the haystack or whether it was shortened.

There are two parameters needed for the automatic evaluation of a ShingleCloud: The *minimal number of consecutive ones (mno)* needed for a sequence of ones in the ShingleCloud to count as a match and the *maximal number of zeroes (mnz)* that may exist within a match without it being interrupted. The second parameter only has an effect when a match has been detected in the first place. Usually the value of this parameter is greater than the shingle size, as a difference in one token would result in *n* non-matching shingles (where *n* is the shingle size). Hence if *mnz=n+1* ShingleCloud would allow for two consecutive unmatched tokens without splitting up a match.

A decision regarding the detection of overlap can be found in different ways: Either by judging if a sufficiently large part of the needle is present in the haystack, i.e. if a consecutive sequence of ones of the given minimal length exists in the ShingleCloud, by means of a containment measure or by a combination of both. When the containment exceeds a given threshold the needle is assumed to be present within the haystack. This containment measure is calculated by dividing the weighted overall number of ones in the ShingleCloud by the length of needle and haystack respectively. The weighted length of ones in the ShingleCloud is calculated as follows: If a one is preceded by a zero, the shingle size is added to the weighted length, otherwise 1 is added. Thus it is not the number of matching shingles that is

used for the calculation of the containment but the number of matching tokens which can be significantly different, depending on the parameters chosen and the distribution of matches within the ShingleCloud.

The ShingleCloud concept is very flexible. In the course of the project we have extended the simple ShingleCloud as described above to include marker shingles that allow for fast lookup of the paragraph in which a match occurred. Furthermore, they allow matches inside one paragraph to be grouped together which can be used to compute containment scores not on the basis of the entire haystack but on a per paragraph level. We also experimented with wildcard shingles that match anything if encountered in a match (a sequence of "ones"). As several words or characters were illegible in the original manuscript we had hoped to improve the initial matching created by ShingleCloud. However, at least for the Chronicles the differences were not significant.

For a thorough documentation and download of ShingleCloud see [Mittelbach, 2010].

3.4 Evaluation

During the course of the *Holinshed Project* all the ~20,000 paragraphs (approximately 2.5 million words in the 1577 edition and almost 4 million words in the 1587 edition) of the 1577 and 1587 versions of Holinshed's Chronicles have been inspected manually. An initial matching was created using the ShingleCloud algorithm as described above which was then inspected and refined by a domain expert. Each paragraph in the 1577 version has been annotated regarding whether this paragraph still exists in the 1587 version and in which paragraph(s) of the 1587 version its content can be found. Thus an annotation set has been created that can serve as gold standard for the evaluation of approaches for automatic detection of overlapping textual paragraphs.

We have split the corpus into several parts using two of them for the evaluation presented here. This had to be done due to the poor runtime performance of GST, which has a worst case complexity of $O(n^3)$, while ShingleCloud and n-gram-overlap run in linear time. Holinshed's Chronicles are internally split up into several books. Part one of the evaluation corpus consists of the first four books plus the preface while part two consists of book five. The following table gives an overview of the two parts.

	Part 1	**Part 2**
Books	Preface, Book 1-4	Book 5
Paragraphs in 1577 version	3135	4476
Words in 1577 version	454,662	554,311
Paragraphs in 1587 version	2685	4377
Words in 1587 version	489,881	1,001,724
Number of matches in annotation	3384	4720
Theoretically possible Matches	$8{,}4*10^6$	$19{,}6 * 10^6$

Table 1: Characteristics of subsets used for evaluation

Since it is possible that a paragraph of the 1577 version has been split up and distributed over several paragraphs in the 1587 version there are possibly more matches in the annotation set than paragraphs in the source version. Due to the high number of possible combinations the number of possibly matching paragraphs has to be narrowed down. Thus the evaluation process is done in two steps:

1. Each algorithm selects a set of candidate pairs of paragraphs. The algorithm selects a pair when one of the containment measures is >0.02.
2. The problem is then transformed into a categorization problem with two categories (hit and miss) using a ten fold cross validation approach based on the given annotations with the containment measures of the paragraph pairs as features. An optimal threshold is computed on the training set (using a brute force approach) which is then used as discriminant to categorize the test set. To achieve high confidence the ten fold is run 10 times.

Now quality measures like the F1 measure, precision and recall can be calculated to compare the different algorithms.

As expected, the LCS algorithm did perform significantly worse on the given corpus than ShingleCloud, n-gram-overlap, or GST; the LCS algorithm reached values for F1 lower than 0.50. Thus the results for LCS are not included in the following. Table 2 shows the F1 measures as well as precision and recall on both parts for n-gram-overlap, GST, and ShingleCloud (SC) in comparison.

Approach	**F1 (σ) Pt. 1**	**Precision / Recall (Pt. 1)**	**F1 (σ) Pt. 2**	**Precision / Recall (Pt. 2)**
GST (MTL = 7)	0,9665 (0,0005)	0,9763 / 0,9577	0,9752 (0,0004)	0,9827 / 0,9681
GST (MTL = 5)	0,9568 (0,0020)	0,9626 / 0,9518	0,9819 (0,0003)	0,9835 / 0,9815
GST (MTL = 6)	0,9669 (0,0008)	0,9742 / 0,9638	0,9794 (0,0004)	0,9827 / 0,9770
SC (n=3, mno=3, mnz=3)	0,9803 (0,0003)	0,9851 / 0,9748	0,9841 (0,0004)	0,9838 / 0,9843
SC (n=3, mno=3, mnz=0)	0,9798 (0,0002)	0,9872 / 0,9721	0,9844 (0,0003)	0,9844 / 0,9845
SC (n=5, mno=1, mnz=0)	0,9713 (0,0005)	0,9730 / 0,9694	0,9853 (0,0003)	0,9867 / 0,9843
SC (n=4, mno=3, mnz=4)	0,9751 (0,0005)	0,9826 / 0,9671	0,9854 (0,0002)	0,9884 / 0,9826
n-gram-overlap (n=5)	0,9637 (0,0008)	0,9659 / 0,9638	0,9854 (0,0001)	0,9880 / 0,9830
n-gram-overlap (n=6)	0,9721 (0,0003)	0,9806 / 0,9638	0,9820 (0,0006)	0,9860 / 0,9792
n-gram-overlap (n=4)	0,9663 (0,0004)	0,9700 / 0,9633	0,9847 (0,0002)	0,9865 / 0,9830

Table 2: Evaluation results

The results show, that all of the algorithms perform similarly well on the given corpus with slight advantages for ShingleCloud over n-gram-overlap and GST, specifically on part one. However, as for most true matches (those that were confirmed by a domain expert) the containment measures were quite high – usually above 33% – and the optimal threshold computed during the evaluation was around 5-10%, smaller differences in the containment measures between the algorithms are not reflected in the results. Figure 1 shows a visualization of the evaluation results on both parts of the Chronicles.

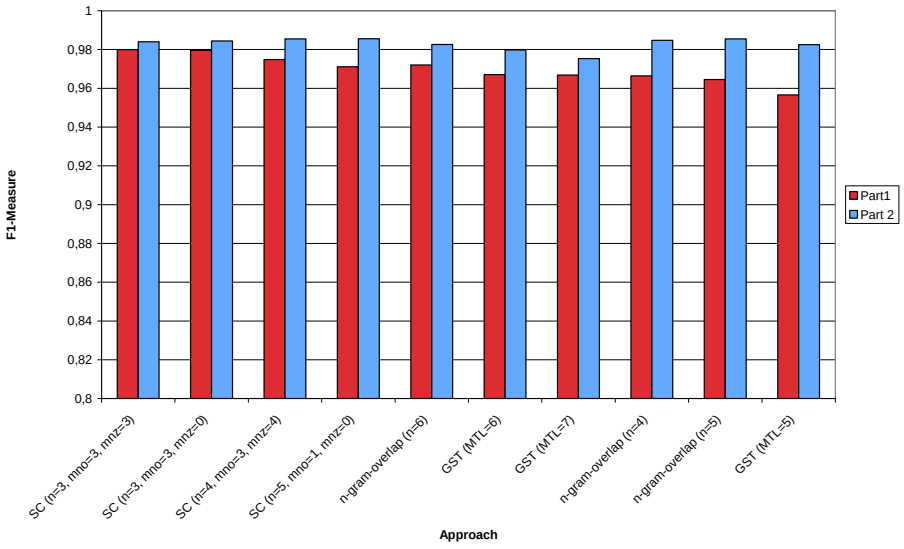

Figure 1: Evaluation Results in Comparison

Since ShingleCloud is based on n-gram-overlap it is expected that both perform almost equally well. However, ShingleCloud has several advantages, including the built-in localisation of matches, the small amount of storage it needs and its runtime efficiency in needle-haystack scenarios. As the ShingleCloud approach is very flexible (see Section 3.3) it can easily be adapted to work in different scenarios; it can be fine-tuned to a certain use-case or run with a one-fits-all configuration. The huge disadvantage of GST is its runtime complexity. While ShingleCloud when run over the entire Chronicles is done in less than three hours, GST needs days if not weeks for this task without producing better results.

4 Visualisation of Overlap - The TEI Comparator

There are some approaches that deal with the visualisation of overlap. In [Klerkx, 2006] relations resulting from the reuse of PowerPoint slides are visualised in various ways to show the form and extent of reuse within the ARIADNE repository.

Visualisation methods like dotplots or Patterngrams are used in the area of plagiarism detection to determine whether there are overlapping sequences between documents or not [Clough, 2001]. Plagiarism detection tools like docoloc use a rather pragmatic method of visualisation and simply add linked annotations to the overlapping text passages. [Johnson, 1994] uses Hasse diagrams to visualise redundancies in legacy source.

The TEI Comparator is a database backed web application that was originally developed as frontend for the *Holinshed Project* and is designed for the visualisation of overlapping passages in two XML files. It was created to allow domain experts to browse through, revise and annotate the initial matching created by ShingleCloud. It works on items of a paragraph-like granularity and allows detecting of how these have been changed from one document to the next. Two matching items can be linked (i.e. marked as matching) and users can put notes on either items or on links. An initial matching (propositions for matching paragraphs) can be created, as a starting point for a more in depth comparison. In case of the *Holinshed Project* the initial matching achieved an accuracy of more than 90% using an early version of the ShingleCloud algorithm (see Section 3.4).

Figure 1 shows the user interface of the TEI comparator. The user can choose which of the two input documents is seen as source document and can then select a "chapter" from the source document to work on. A table of contents, to select the chapter from is automatically generated from the underlying XML structure. The work unit is then displayed in the left column allowing the user to browse through its paragraphs. For each of the paragraphs, the right hand column shows linked paragraphs in the target document and allows the user to confirm links created by the initial matching process or further refine them (remove matches or search for new matches either with ShingleCloud, via the document's structure or via full text search). The matching portions of two paragraphs can be highlighted.

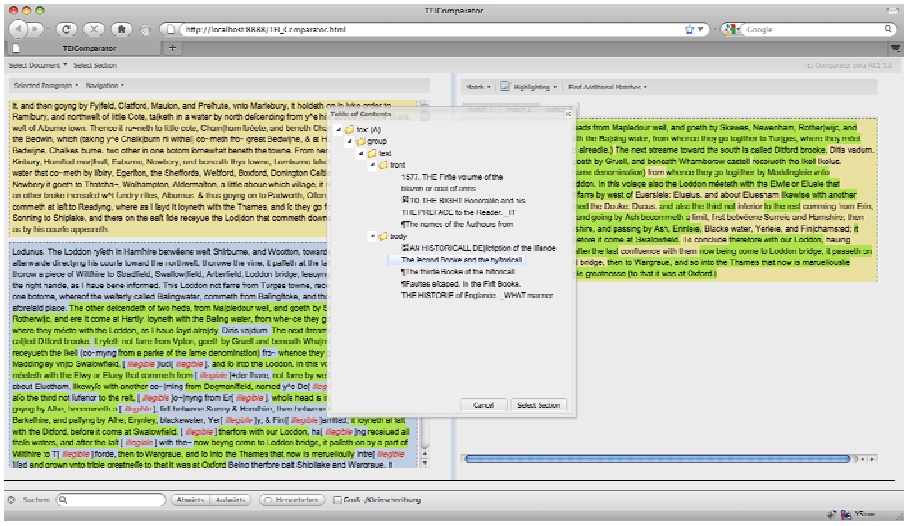

Figure 2: The TEI Comparator GUI

The resulting data (matches and notes) can be exported into a TEI-marked-up XML file [TEI, 2010], which can then be further processed with standard XML tools (see for example http://www.english.ox.ac.uk/holinshed/). For documentation and download see [Mittelbach, 2010b].

5 Conclusions and Future Work

In this paper we have proposed an algorithm and tool to support the tracking of information flow. Specifically in the context of historical documents this can help greatly to understand and interpret events and changes in attitudes and the political or religious landscape of those times. The evaluation has shown that the proposed ShingleCloud algorithm is better suited for the task of overlap detection task than state of the art approaches. Evaluations we have done on other corpora have confirmed this experience. Besides its use in the *Holinshed Project* the proposed system can be used in arbitrary scenarios where textual overlap needs to be detected and visualised in an effective and fast manner. Future work in this area might include the adaptation of the tool and algorithm for slightly different use cases like comparison of and search in DNA sequences or functional extensions to the given UI.

References

[Broder, 1997] Broder, A. Z. (1997), On the Resemblance and Containment of Documents, in 'SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997', IEEE Computer Society, Washington, DC, USA, pp. 21.

[Burrows, 2007] Burrows, S.; Tahaghoghi, S. M. M. & Zobel, J. (2007), 'Efficient plagiarism detection for large code repositories', Softw. Pract. Exper. 37(2), 151--175.

[Clough, 2001] Clough, P. (2001). Measuring Text Reuse and Document Derivation. Postgraduate transfer report, Department of Computer Science, University of Sheffield, UK.

[Clough, 2003] Clough, P. (2003), 'Old and new challenges in automatic plagiarism detection', Online: http://ir.shef.ac.uk/cloughie/papers/pas-plagiarism.pdf, (zuletzt abgerufen am 16.11.2009).

[Hamid, 2009] Hamid, O. A.; Behzadi, B.; Christoph, S. & Henzinger, M. (2009), Detecting the origin of text segments efficiently, in 'WWW '09: Proceedings of the 18th international conference on World wide web', ACM, New York, NY, USA, pp. 61--70.

[Holinshed, 2010] Holinshed's Chronicles, Online, http://www.cems.ox.ac.uk/holinshed/

[Jaccard, 1901] Jaccard, P., 'Étude comparative de la distribution florale dans une portion des Alpes et des Jura', Bulletin del la Société Vaudoise des Sciences Naturelles 37, 547--579.

[Johnson, 1994] Johnson, J. H. (1994), Visualizing textual redundancy in legacy source, *in* 'CASCON '94: Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, , pp. 32.

[Kim, 2009] Kim, J. W.; Candan, K. S. & Tatemura, J. (2009), Efficient Overlap and Content Reuse Detection in Blogs and Online News Articles, *in* '18th International World Wide Web Conference'.

[Klerkx, 2006] Klerkx, J.; Verbert, K. & Duval, E. (2006), Visualizing Reuse: More than Meets the Eye, in 'Proceedings of the 6th International Conference on Knowledge Management (I-KNOW) 2006', pp. 489-497.

[Loose, 2008] Loose, F.; Becker, S.; Potthast, M. & Stein, B. (2008), Retrieval-Technologien für die Plagiaterkennung in Programmen, in Joachim Baumeister & Martin Atzmüller, ed.,'Proceedings of the Information Retrieval Workshop at LWA 2008', University of Würzburg, Germany, , pp. 5-12.

[Lyon, 2001] Lyon, C.; Malcolm, J. & Dickerson, B. (2001), Detecting Short Passages of Similar Text in Large Document Collections, in Lillian Lee & Donna Harman, ed.,'Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing', pp. 118--125.

[Maier, 1978] Maier, D. (1978), 'The Complexity of Some Problems on Subsequences and Supersequences', J. ACM 25(2), 322--336.

[Metzler, 2005] Metzler, D.; Bernstein, Y.; Croft, B. W.; Moffat, A. & Zobel, J. (2005), Similarity measures for tracking information flow, in 'CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management', ACM, New York, NY, USA, pp. 517--524.

[Mittelbach, 2010] Mittelbach, A. & Lehmann, L. (2010), 'ShingleCloud Library for approximate string matching', Online: http://www.kom.tu-darmstadt.de/en/downloads/software/shinglecloud/, Zuletzt abgerufen am 13.01.2010.

[Mittelbach, 2010b] Mittelbach, A. & Cummings, J. (2010), 'TEI-Comparator, Online: http://tei-comparator.sourceforge.net/, Zuletzt abgerufen am 03.03.2010.

[TEI, 2010] TEI Consortium, eds. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. 1.6.0. Last updated on February 12th 2010.. TEI Consortium. http://www.tei-c.org/Guidelines/P5/ (last accessed 01.06.2010).

[Wise, 1993] Wise, M. J. (1993), 'Running Karp-Rabin Matching and Greedy String Tiling', Technical report, Basser Department of Computer Science - The University of Sydney.