Pedigree Tracking in the Face of Ancillary Content

Eugene R. Creswick and **Emi Fujioka** and **Terrance Goan**[1]

Abstract. The accurate tracking and retrieval of content pedigree is a quickly growing requirement as our abilities to create information assets increases exponentially. Plagiarism detection, accurate accreditation, and classification tasks all rely on the ability to determine where content is being used and where it originated. We present an approach to document pedigree tracking that is based on an efficient disk-based data structure and the use of two contrasting collections of historical text. These collections enable content of two types (or degrees of importance) to be defined and accounted for when locating documents with overlapping content. This approach is resilient in the face of substantial ancillary content and paraphrasing, two common sources of error in existing content tracking techniques.

1 INTRODUCTION

It has become clear that our ability to create vast information assets far outstrips our ability to exploit and protect them. The accurate tracking of information use and reuse in documents and on the Web is important for many reasons such as detecting unauthorized use, and properly tracking citations during an authoring session. One particularly challenging application arises in the world's intelligence communities as they seek to: improve information awareness amongst analysts; improve the reliability of intelligence; safely share information with warfighters and allies; and root out malicious insiders. One means to mitigating these challenges is to provide reliable knowledge of the provenance (or ideally, pedigree) of documents.

This knowledge would allow, for instance, analysts to identify the source of information underpinning an intelligence report. Once the provenance of a document (or portion of a document) is known, that knowledge can be used to: create and enforce classification policy rules at a given site, narrow the scope of a search for information leaks, and enhance the authoring and reading processes by automatically presenting references to related documents.

There are two primary approaches to establishing information provenance. First, we might seek to develop information systems or processes that utilize meta-data to track the source of data imported into new intelligence products. Unfortunately, the wide variety of information systems makes such an approach impractical, and the adoption of techniques that would allow existing systems to easily communicate is highly unlikely.

The more attractive alternative is to recover provenance knowledge on-demand, as required by users. This approach is exemplified by plagiarism detection tools. These tools apply various methods of document similarity detection to determine when content has been reused; however, all of the approaches we are aware of can be biased by the presence of common, inconsequential text (often termed *boilerplate*—content that has relatively little semantic meaning compared to its context).

[1] Stottler Henke Associates, Inc., email: rcreswick@stottlerhenke.com

Content is reused for various purposes in many domains—not all of which involve malicious intent. For instance, internal communications and documents such as whitepapers and grant proposals commonly reuse sections of text. Frequently reused sections include the company descriptions, key personnel (resumés), introductions and portions of related work. Furthermore, the formatting of documents within a company is generally standardized: all documents of a given type often share fixed section headings, page headers and page footers. These duplicated portions may or may not constitute a meaningful link between two given documents. In many cases, all of these areas will be considered boilerplate, while in other situations some of these duplicated areas may be of consequence. Headers and footers, for example, are probably always boilerplate; however, the same cannot be said of introductions.

We present a novel approach to pedigree tracking in which content can be marked as either *open* or *sensitive*. *Open* content is considered to be inconsequential, and will not be incorporated in the overlapping content that is used to determine a document's pedigree. All other content is marked as *sensitive*.

We begin with a discussion of the related work in plagiarism detection, information provenance, and content tracking in Section 2. In Section 3, we present the InfoTracker algorithm for document pedigree tracking. Section 4 describes our initial evaluation of InfoTracker on the document pedigree task. Finally, we summarize our findings and discuss directions for future work in Section 5.

2 RELATED WORK

Metzler, et al. present five levels of similarity used to measure the relationship between two portions of text and they examine different methods of comparing sentences and documents [8]. The levels considered cover the range of overlapping semantics to exact duplication. A measure of similarly with a similar purpose is Levenshtein distance—a method that calculates the number of modifications to a string needed to transform it into another given string. This technique provides a quantitative measure of the difference between any two arbitrary strings in quadratic time. This is well suited to applications that involve short strings, such as spell checking individual words in a document. As is the case with the techniques presented by Metzler, et al. Levenshtein distance is a direct pairwise comparison of two fixed strings. While such a gamut of comparisons is well suited to determining if two portions of text are related, the pairwise comparisons required to determine their similarity is prohibitively expensive when the corpus of documents to search reaches the hundreds or thousands.

Jagadish, et al. address the topic of similarity-based queries, in which the results are based on similarity rather than exact match [7]. While their approach is able to detect strings that are not exact duplicates, it is only able to do so for types of similarity that have been

exactly specified in advance. The approach is also targeted at comparing individual characters (or more generally, symbols) for similarity (‘a’ vs. ‘â’, for example). This may be a worthy addition to our approach, as it could ease the handling of content that includes Unicode characters or the results of optical character recognition (OCR).

Eppstein, et al. present methods for fast sequence alignment in the context of RNA analysis [3]. Their approach is similar to the fragment-based approach presented by Wilbur and Lipman [11]. There are similarities between using text fragments and suffixes, but both Eppstein, et al. as well as Wilbur and Lipman’s approaches require pairwise comparisons, whereas our approach uses an index of the historical content to speed retrieval.

Fingerprinting, as presented by Hoad and Zobel [6], hashes select substrings and compares the aggregated results for each document. Unfortunately this approach is highly sensitive to the choice of substrings. Depending on the selection approach used (of which there are myriad), the fingerprints for two documents may be widely divergent while the documents differ only by a few insertions or deletions. Stein presents a similar approach to fingerprinting termed “fuzzy fingerprints” that calculates a hash of a document based on the distribution of common prefixes [9]. This hash is calculated in a way that “fuzzifies” the result, increasing the chances that similar documents will have the same hash. Hash collisions are then indicative of document similarity. Vector Space models have also been shown to be successful, and in the case of Hoad and Zobel, such approaches outperformed the fingerprinting approach in all respects [6].

The commercial TurnItIn [1] plagiarism detection service utilizes a different approach—it relies on detecting long strings of words shared by co-derived documents. Regrettably, this tactic is susceptible to false-negative errors when faced with heavily edited text (edits reduce the chance that long strings will match) and false-positive errors when faced with shared but inconsequential text. Indeed, such boilerplate content can sway all of the plagiarism detection techniques that the authors are aware of.

Eissen, et al. present a method of using suffix trees for document similarity that is very similar to our InfoTracker approach [2]. Eissen, et al. construct a suffix tree from two documents and calculate similarity based on the edges that are shared between the two documents in the tree. This approach differs from ours in one fundamental way: Eissen, et al. do not incorporate any filtering content (such as boilerplate) into the similarity calculations.

3 ALGORITHM DESCRIPTION

We have developed a new approach to document pedigree tracking and implemented a prototype—termed InfoTracker—in order to support new document pedigree tracking applications and to overcome the shortcomings of past techniques.

Fundamental to our InfoTracker system is the concept of a suffix tree [10], which allows for fast indexing and querying of the entire content of indexed text. Our contribution is in the development of a means to detect derivative text (and thereby information provenance) in the presence of substantial ancillary content.

InfoTracker accomplishes this by contrasting two distinct collections of text, one composed of content of interest (eg: confidential information in intelligence reports) and the other composed of benign or “uninteresting” text. Some examples of “uninteresting” text are publicly available (and therefore unclassified) web documents, open-source news articles, document headers, footers and other common text. The prior collection of content represents the initial *sensitive* collection, while the later collection of content represents the initial *open* collection. These collections also need not be in separate documents, rather, parts of a document can be considered *sensitive* while other portions are considered *open*.

While the suffix tree is a very strong foundation upon which to develop applications seeking to detect overlapping content, it is not in and of itself sufficient to deliver the capabilities described above. For instance, employed naively, these data structures may generate very high false alarm rates due to the inconsequential overlaps amongst documents that result from institutional boilerplate, document structure (e.g., “Figure 1”, page headers and footers, etc.), or common figures of speech. We have addressed this problem by devising a novel extension to the suffix trees that allows us to contrast different corpora. In particular, our approach utilizes a large randomly selected collection of texts to identify those common word sequences that may occur by chance and are therefore not useful in determining co-derived text. Consider the example in Figure 1, which shows a simplified representation of the text index. The suffix tree stores all suffixes of the sequence of text representing a document. The figure shows how we can utilize information about the source of text to identify those sequences of characters/words that appear unique to the documents with content we wish to track. In this example, InfoTracker would find that any document containing either “as their hideout” or “hotel as their hideout” shared a common source with document s1. The content of Figure 1 could be drawn from multiple sources: the benign terms may originate in newspapers, while the sensitive text may be found in an internal report (for example).

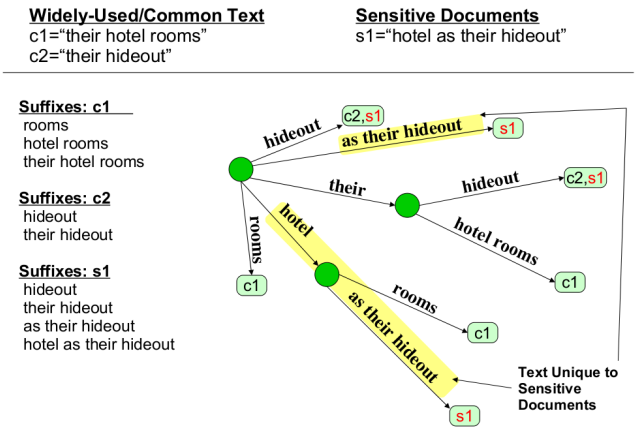

Figure 1. A simplified visualization of the suffix tree built for a small sample of content. The storage details have been omitted for clarity.

InfoTracker is able to accurately judge the likelihood that text strings could be reproduced independently by analyzing string overlaps in the light of general language usage. In other words, while previous approaches exploit the rareness of long strings, InfoTracker exploits its greater knowledge of common text patterns to recognize much shorter strings of text that are likely to be derived from other historical documents. This allows InfoTracker to succeed in a number of situations where past approaches may fail, including identification of co-derivative relationships between paraphrases or between documents processed with different OCR systems (each of which generates its own errors).

InfoTracker indexes the two collections (of *open* and *sensitive* text) with an implementation of a suffix tree based on a String B-Tree (The use of a String B-Tree enables processing of document

collections that will not fit in memory [4].) When building the suffix tree, InfoTracker keeps track of the source of each suffix and whether that suffix was found in a section of content marked as *open*. It does this by annotating the leaves of the suffix tree with references to the source documents and a Boolean flag that indicates whether the suffix is *open* or *sensitive*.

When a query document is submitted to InfoTracker, a fixed-sized window is initialized at the start of the query text. (We have found that windows in the range of 40-80 characters work well.) The root of the suffix tree is checked for an edge labeled with the first token in the window. If the token is found, that token is consumed and the search continues from the resulting node in the suffix tree with the next token in the window. This continues until the suffix tree is no longer able to match tokens in the document or the end of the query is reached. If the matched text is *not* longer than the size of the window, the text is discarded. If the matched text *is* longer than the window, then the index and length of that text in the query document is stored along with the *open* or *sensitive* state of the overlap and a pointer to the overlapping document. In either case, the window is then shifted forward one token, and the search repeats from the root of the suffix tree. This procedure iterates across the entire query document, collecting each overlapping sub-string that is longer than the window size.

Once these overlaps have been collected, the set of *open* regions of text are subtracted from the set of *sensitive* regions of text. For example, if a *sensitive* overlap spans from index 0 to index 100, and an *open* overlap spans from index 50 to index 120, then the result is a *sensitive* overlap from index 0 to index 49. The remaining *sensitive* overlaps are filtered once again to remove any that are now shorter than the window length.

Each of the resulting overlaps has a pointer to a document that contains sensitive content used in the query document. However, in our experience, this list of source documents is still very large (106–233 source documents were found for each query in our experiment). Some of the overlapping portions of text are the result of an incomplete *open* collection, and should be filtered further. The principle idea is that the overlapping portions of text that appear in many documents (eg: section titles, like "EVALUATION") are of less interest than portions of text that appear in few documents (such as content about specific methodologies). To leverage this, we have defined a measure of inverse overlap frequency (IOF) for each overlapping section of text.

For a given overlap, all of the documents that contain that content are collected and aligned based on the location of the overlapping text in the query document. For example, if the phrase "an ontology" occurs in two historical documents, but one of the documents contains additional duplicated text: "an ontology improved", then the matching portions of the overlapping phrases are aligned[2]:

```
Overlap 1: an ontology
Overlap 2: an ontology improved
```

The overlap frequency is then calculated for each character index in the overlapping text. In this example, the overlap frequencies are:

```
Overlap 1: an ontology
Overlap 2: an ontology improved
OF:        22222222222111111111
```

Note that the "o" in "improved" has an overlap frequency of 1 because there is no corresponding character at that index in Overlap 1.

[2] The example strings used here have been shortened to reduce the complexity of the example. The actual strings detected are somewhat longer.

The overlap frequency indicates how common a given portion of text is, but we are more interested in the content that is rare. Therefore we simply invert the overlap frequency directly, as shown in Equation (1). In the example above, each of the characters in " improved" have an OF of 1. The characters in "an ontology" each have an OF of 2.

$$\text{iof}_i = \frac{1}{\text{overlaps containing character at } i} \tag{1}$$

Equation (2) shows the IOF calculation for the first character of overlap 2:

$$\text{iof}_0 = \frac{1}{2} = 0.5 \tag{2}$$

The inverse overlap frequency values are then summed to generate a length-IOF score for that overlapping region. The additional content in the second overlap (" improved") greatly increases the effect of that overlap on the ranking of these documents, as can be seen in the respective length-IOF scores:

```
length-IOF(Overlap 1):  5.5
length-IOF(Overlap 2): 14.5
```

The overlapping regions from each historical document are scored in this way, and the scores summed to generate an overall rank for that historical document. Therefore, common overlapping sections are considered important if they are large, while shorter common overlapping sections have much less influence. This has the added benefit that the large sections which are common are more evident when users are viewing results, and these regions can more easily be marked as open, if indeed they are inconsequential.

4 EVALUATION

The InfoTracker prototype generates a list of documents that are deemed similar to the query document. To obtain an initial gauge of the performance of this approach, we have conducted an exploratory experiment. The performance of the InfoTracker prototype is evaluated with precision/recall measurements, as generally used for search tasks. A vector-space approach using a cosine similarity metric was used to provide a point of comparison, since the vector-space approach is well known and understood. The vector-space implementation uses a typical bag-of-words with TF–IDF weights. No stop words were used in either approach.

4.1 Experimental Data

Our evaluation uses a corpus of 272 proposals prepared by a single company between January 1st, 2000 and December 31st, 2007[3]. These documents share considerable content in the form of personnel resumés, facilities descriptions, related work, and smaller portions of the techniques shared by proposals in similar technological areas. Additionally, each proposal uses the same document template and has nearly identical headers, footers, section headings and other ancillary text.

This corpus presents two real-world challenges to plagiarism detection and information provenance. Much of the duplicated content is gradually updated over the years from proposal to proposal. This introduces hundreds of minor alterations as authors fix typos, introduce new typos, specialize content for the topic at hand, rename old

[3] These documents range in size from 44.0 kilobytes to 111.6 kilobytes ($\mu = 80.0$, $\sigma = 11.9$).

projects, or introduce new project names into sections that are largely boilerplate. Each of these changes breaks a previously contiguous section of duplicated content into smaller pieces. Furthermore, many portions of content that are duplicated are of no interest or concern whatsoever. One can imagine that the only "secrets" to be protected in this corpus are directly related to the technologies that were proposed over the years. The collection of resumés and shared related work sections are typically benign, and would be safe for public release. These benign sections should therefore not influence the selection of source documents when identifying the pedigree of a proposal in order to inform the classification of that proposal's content.

4.2 Experiment Description

The proposals in this data set were divided into two groups:

2000-2006 proposals (234 documents): These proposals were loaded as historical documents, with the key personnel, company description, and related work sections automatically marked as *open*[4] and with the remainder of the documents' content marked as *sensitive*. The authors of these proposals indicated that those sections are very rarely subject to substantial change, and it is reasonable to assume that this type of foreknowledge is available to some degree in many real-world scenarios.

2007 proposals (38 documents): The more recently authored proposals were used as a test set. Each proposal was loaded as a query document (in the order they were authored) and the documents returned were recorded as the results. The query document was then added to the InfoTracker tree in the same way as the historical documents. This allows for recent documents that reference proposals written since 2006 to be properly handled.

One of the proposal authors built an oracle by manually examining each of the 38 query documents and comparing each query document with the full collection of 272 proposals to determine which (if any) proposals were sources of significant content. Eight documents were deemed to derive content from no other proposals in the corpus, one document drew content from 23 others, and the remaining 31 documents were arrayed in the intervening range ($\mu = 4.76$, $\sigma = 5.16$).

In our experiment, we initially loaded the suffix tree with a large collection of general text from the web to provide a base *open* collection before indexing the 234 historical documents and processing the query documents. This initial open collection consisted of 590 documents, with a total size of 780MB.

4.3 Results

Both InfoTracker and the vector-space approach generated lists of results for each query that are nearly all-encompassing. Nearly every historical document was included in the InfoTracker results (although many documents have very low scores) and the vector-space approach, by design, simply ranked every document. In order to determine meaningful values for precision and recall we needed to cut off the results list at a shorter, more reasonable length. Because of the number of results in the oracle, we decided to consider the top 23 results for each query for both algorithms. This is the lowest number of results that can possibly have 100% recall. Table 1 shows the results for the two approaches. Note that the precision values are particularly low because most of the query documents have very few true source documents ($\mu = 4.76$) compared to the size of the result list considered (23).

Algorithm	Precision	Recall
Vector Space	0.119	0.764
InfoTracker	0.167	0.913

Table 1. Results for InfoTracker compared to the vector space approach for detecting source documents.

4.4 Trimming Results

The ranked list of results generated by the prototype are generally too large to be of use in an automated system, since most of the results are false positives. In our experimentation this has resulted in lists of 106–232 related documents for a given query. Observation of the results for one query indicate that the ranking scores fit a skewed distribution with a very long flat tail. This tail is made up of documents that only share boilerplate content with the query document.

Table 2 shows the top 15 results for a query, along with the scores for each retrieved document. Notice that the top six results have substantially higher scores than the remainder. One justification for this immediate fall-off of the ranking scores is that the tail consists of documents that share content of no importance (headers, footers, section titles and the like) while the first few results are drawn from a different population of documents that share substantial content with the query document. If this assumption is valid, then the top ranked results should be outliers with respect to the rest of the retrieved results.

Table 2. The top 15 (of 116) results for a document query in the InfoTracker prototype.

Rank	Score	File
1	6289.995	Document-92
2	3206.340	Document-21
3	1630.607	Document-13
4	1366.318	Document-46
5	1157.704	Document-1
6	1103.442	Document-43
7	624.238	Document-114
8	327.533	Document-67
9	273.651	Document-74
10	263.037	Document-48
11	244.407	Document-10
12	238.435	Document-113
13	207.320	Document-101
14	134.991	Document-58
15	131.520	Document-12
...	...	...

The prototype uses a definition of outliers that is based on the inter-quartile range[5] to determine which results to retain and which should be trimmed. Equation (3) shows how the threshold for trimming is calculated:

$$\text{threshold} = Q_3 + (N \times (Q_3 - Q_1)) \tag{3}$$

Q_1 represents the lower end of the inter-quartile range, Q_3 represents the upper end of the inter-quartile range, and N is a constant that

[4] A simple template consisting of regular expressions was used to identify these sections.

[5] The inter-quartile range, or IQR, is the range of values that includes the middle 50% of the data points in a distribution, 25% of the data falls below the IQR, while 25% of the data have values greater than the IQR.

determines the degree of extremity required of the outliers. N can be used to shift the balance between precision and recall. For example, the full 116 data points of the results in Table 2 have a lower quartile of 1.837 (Q_1) and an upper quartile of 47.250 (Q_3), indicating that 29 data points have scores under 1.837 and 87 data points have scores under 47.250. With $N = 6$, the threshold is set to 319.728, and only the top seven results are retained.

The experiment described in Section 4.2 was run with varying values of N from the range [0–6]. Low values of N represent very conservative estimates of the distribution of unrelated documents, and sets a low threshold for outliers. Each full-unit increment increases the threshold by an amount equal to the inter-quartile range, trimming the query results more aggressively. The full test corpus of 38 query documents was run on each successive value of N and the average number of results, average precision, and average recall are recorded in Table 3.

Table 3. Precision/Recall statistics for the pedigree detection experiment, as a function of outlier extremity.

N	Result Count	Precision	Recall
No Trimming	162.53	0.03	0.98
0	40.95	0.11	0.97
0.5	28.71	0.14	0.93
1	22.29	0.16	0.91
1.5	18.92	0.19	0.90
2	15.81	0.21	0.88
2.5	13.47	0.23	0.87
3	11.76	0.24	0.84
3.5	10.50	0.26	0.84
4	9.63	0.27	0.81
4.5	8.82	0.29	0.80
5	8.18	0.31	0.78
5.5	7.55	0.33	0.78
6	7.13	0.36	0.77

Table 3 clearly shows the control available over the balance between precision and recall, and demonstrates the amount of result trimming that can safely be applied for a desired level of recall. Even the most minimal trimming attempted shortened the results list by over 60% (compared to the initial minimum size of 106 results) yet only reduced average recall by 1% compared to the case where no trimming was done.

5 CONCLUSIONS AND FUTURE WORK

During the execution of this project, we have identified a number of directions to pursue in the future:

Evaluate in an Active Learning scenario: Foremost in our future goals is to perform an exhaustive evaluation of the InfoTracker prototype in a scenario that takes advantage of Active Learning to identify and mark boilerplate content while the system is in use.

Incorporate time stamps: The current approach does not take the temporal aspect of document authoring and reuse into account when determining pedigree. Therefore, if a query document shares a source with a historical document, then both the source and the sibling document are likely to be returned in the list of results. These false–positive results can be reduced by considering the dates that the returned documents are authored, possibly presenting the results hierarchically, or only returning either the youngest or oldest sources.

Overlap size: Another indication of the actual structure of the document pedigree is available in the content of the overlapping sections themselves. For example, if document C contains content taken directly from document B, which was originally taken from document A, there is a chance that the overlapping section that C shares with B will be larger than the overlapping section found to be common to C and A. Indeed, it is highly likely that the overlapping content between C and A is a proper subset of the overlaps shared between C and B. In–depth analysis of the similarities between overlapping content shared between multiple documents may reveal more intricacies of the document pedigree.

Alternative outlier definitions: The characteristics of the flat tails of each results list may more closely fit a certain type of distribution. If so, a more complex outlier detection method (such as Grubbs' Test for Outliers [5]) may be able to determine a threshold for result trimming that improves precision.

We have presented an approach to document indexing and search that enables the detection of document pedigree when substantial ancillary content is present. We have compared this approach to the common vector-space approach used frequently for information retrieval tasks, showing that our approach is better able to manage the presence of ancillary content. InfoTracker makes use of efficient disk-based data structures that promise to scale well with large corpora that do not fit in memory; however, a thorough evaluation of the scalability of InfoTracker is still a topic for future investigation.

Evaluation on the proposal data set revealed that a great deal of control is available over the precision/recall trade-off. This can be incorporated into tools in the future to adapt to the needs at hand. For example, applications dealing with the dissemination of potentially classified content will require a high degree of recall, while an application where the emphasis is on immediate results may choose to avoid false positives with higher precision.

REFERENCES

[1] TurnItIn. Website: http://www.turnitin.com, June 2008.

[2] Sven M. Eissen, Benno Stein, and Martin Potthast, 'The suffix tree document model revisited', in *Proc. of the 5th International Conference on Knowledge Management (I-KNOW 05)*, pp. 596–603, Graz, Austria, (July 2005). Know-Center. ISSN 0948-695x.

[3] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Guiseppe F. Italiano, 'Efficient algorithms for sequence analysis', in *SEQS: Sequences '91*, (1991).

[4] Paolo Ferragina and Roberto Grossi, 'The string b-tree: a new data structure for string search in external memory and its applications', *J. ACM*, **46**(2), 236–280, (March 1999).

[5] Frank E. Grubbs, 'Procedures for detecting outlying observations in samples', *Technometrics*, **11**(1), 1–21, (February 1969).

[6] Timothy C. Hoad and Justin Zobel, 'Methods for identifying versioned and plagiarized documents', *Journal of the American Society for Information Science and Technology*, **54**(3), 203–215, (2003).

[7] H. V. Jagadish, Alberto O. Mendelzon, and Tova Milo, 'Similarity-based queries', in *PODS '95: Proc. of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 36–45, New York, NY, USA, (1995). ACM.

[8] Donald Metzler, Yaniv Bernstein, Bruce W. Croft, Alistair Moffat, and Justin Zobel, 'Similarity measures for tracking information flow', in *CIKM '05: Proc. of the 14th ACM international conference on Information and knowledge management*, pp. 517–524, New York, NY, USA, (2005). ACM.

[9] Benno Stein, 'Fuzzy-fingerprints for text-based information retrieval', in *Proc. of the 5th International Conference on Knowledge Management (I-KNOW 05)*, pp. 572–579, Graz, Austria, (July 2005). Know-Center. ISSN 0948-695x.

[10] Esko Ukkonen, 'On-line construction of suffix trees', *Algorithmica*, **14**(3), 249–260, (1995).

[11] W. J. Wilbur and David J. Lipman, 'Rapid similarity searches of nucleic acid and protein data banks', *PNAS*, **80**(3), 726–730, (February 1983).