Engineering Applications of Artificial Intelligence 64 (2017) 261–271

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Docode 5: Building a real-world plagiarism detection system

Gaspar Pizarro V., Juan D. Velásquez *

Universidad de Chile, Chile

ARTICLE INFO

Keywords:
Plagiarism detection
Software engineering
MapReduce

ABSTRACT

Plagiarism refers to the appropriation of someone else's ideas and expression. Its ubiquity makes it necessary to counter it, and invites the development of commercial systems to do so. In this document we introduce Docode 5, a system for plagiarism detection that can perform analyses on the World Wide Web and on user-defined collections, and can be used as a decision support system. Our contribution in this document is to present this system in all its range of components, from the algorithms used in it to the user interfaces, and the issues with deployment on a commercial scale at an algorithmic and architectural level. We ran performance tests on the plagiarism detection algorithm showing an acceptable performance from an academic and commercial point of view, and load tests on the deployed system, showing that we can benefit from a distributed deployment. With this, we conclude we can adapt algorithms made for small-scale plagiarism detection to a commercial-scale system.

1. Introduction

Plagiarism is defined as the unauthorized use or close imitation of the ideas and language expression of someone else and involves representing their work as one's own (Hannabuss, 2001). As an example of how pervasive it can be, a study (Molina et al., 2011) found that approximately fifty percent of students in Chile have taken information from external sources without citation, making it necessary to take measures for countering it. Since detection of human plagiarism is labor-intensive (Kakkonen and Mozgovoy, 2010), this problem has pushed efforts to automate plagiarism detection. Thus, there have been efforts both in academic research and commercial systems for detecting plagiarism. However, in each area the main focus is different; in academic research, the focus is on algorithms and systems for detecting sophisticated plagiarism cases, as can be seen in Potthast et al. (2009), whereas in commercial systems, the focus is on user interfaces and their impact on the process of writing good documents. For instance, the plagiarism detection function in Turnitin (0000) is just one of their many functionalities for helping to write good documents, and their interfaces are polished to the end-user.

Our objective is to deploy a commercial plagiarism detection system. When deploying on a commercial scale, there are some problems related to massive use. These problems are related to the analysis of large document collections and the fact that a plagiarism detection system is meant to download documents, interacting with the World Wide Web. So, in this paper we present Docode 5, a commercial plagiarism detection system. We present Docode 5 from end to end, from the algorithms used up to the user interfaces, and since we are aiming for a commercial deployment, we show the measures we take for dealing with the problems described above.

The rest of this document is structured as follows. Section 2 contains a brief explanation and survey of the subjects this paper covers, namely plagiarism detection, algorithms for dealing with large document collections and plagiarism detection systems. Section 3 explains the functionalities offered by the system and the algorithms used in it. Section 4 shows the logical architecture of the deployed system and the technologies used for deploying it. Section 5 shows some tests we ran with the system to prove its performance. Section 6 shows what can be improved in this system as a future work, and, finally, Section 7 shows the conclusions from this work.

2. Related work

2.1. Plagiarism detection

A first survey on the subject can be found in Clough et al. (2003), where the author discusses the forms of plagiarism, the difference between plagiarism in natural language documents and in structured text documents, namely computer code. An initiative to push forward

* Corresponding author.
E-mail addresses: gaspar.pizarro.v@wic.uchile.cl (G. Pizarro V), jvelasqu@dii.uchile.cl (J.D. Velásquez).

http://dx.doi.org/10.1016/j.engappai.2017.06.001
Received 29 July 2016; Received in revised form 6 April 2017; Accepted 2 June 2017
Available online 20 July 2017
0952-1976/

research in automatic plagiarism detection was done in the PAN competition from 2009 to 2014 (Potthast et al., 2009, 2010a; Stein et al., 2011; Potthast et al., 2012, 2013, 2014; Hagen et al., 0000), which delivered many different approaches to the task, and also delivered datasets and an evaluation framework that can be used to compare different plagiarism detection strategies (Potthast et al., 2010b). Prior to 2013, the idea of plagiarism detection was done on a dataset with an "all-versus-all" approach, that is, given two sets of documents, the *suspicious* and the *source* set, the task was finding all pairs of source and suspect which have plagiarism. From 2013 onwards, the task was focused on plagiarism detection on the web, and decomposed into two subtasks:

Source retrieval Given a suspicious document, the task is to find similar documents in a collection only accessible through a search engine (provided by the competition). The goal in this subtask is to find the documents that are likely to have plagiarized sections with the suspicious one, while minimizing the number of queries to the search engine and the number of downloaded documents.

Text alignment Given a pair of documents, to find the plagiarized sections in common between them. The goal here is what is usually known as plagiarism detection, that is, to find only the plagiarized sections while coping with obfuscation techniques like paraphrasing, translation[1] or summarization.

The task of plagiarism detection is also called *external* plagiarism detection, because a suspicious document is analyzed against a source document, or against a group of source documents. This is opposed to *intrinsic* plagiarism detection, where the task is to detect plagiarized passages of text in a suspicious document without a reference source, that is, to find text that "looks" too different to the document to be written by the same author of the rest of the document. This work is focused on external plagiarism detection, so when we refer to plagiarism detection, unless stated otherwise, we mean *external* plagiarism detection.

Formally, as described by Stein et al. (2011), a plagiarism case between two documents is described as a quartet $s = (s_{plg}, d_{plg}, s_{src}, d_{src})$, where a section s_{plg} of a suspicious document d_{plg} is a plagiarized version of a section s_{src} of a source d_{src}. Similarly, a detection $r = (r_{plg}, d_{plg}, r_{src}, d'_{src})$, is where a discovered section r_{plg} of the suspicious document is regarded as plagiarized from a section r_{src} from a document d'_{src}. With this framework in mind, we can consider the source retrieval task as finding the right source document, that is, $d'_{src} = d_{src}$, while the text alignment task can be described as finding the right sections in the suspicious document, that is, $s_{plg} \approx r_{plg}$. Focusing on the text alignment subtask, we say that a plagiarism detection algorithm is evaluated with a set S of plagiarism cases (all with the same d_{src} and d_{plg}), and yields a set R of detections. The quality of an algorithm can be evaluated as an Information Retrieval task, with a modified version of precision and recall (Potthast et al., 2010b). These measures treat contiguous plagiarized sections as basic retrieval units, as opposed to treating characters as basic retrieval units, for which we can use the standard definitions of precision and recall (Potthast et al., 2009). With the definitions of plagiarism case and detections shown before we can define the performance measures as follows:

Precision This measure is the average fraction of each detection that is indeed a plagiarism case.

$$precision(S, R) = \frac{1}{|R|} \sum_{r \in R} \frac{\left|\bigcup_{s \in S}(s \sqcap r)\right|}{|r|} \tag{1}$$

where $s \sqcap r$ is the set of overlapping characters between the case s and the detection r. We say that r *detects* s if $s \sqcap r \neq \emptyset$.

[1] More exactly, this is using a translation engine to translate a document to a different language, and then translating the translated document back to the original language, which introduces differences in the document.

Recall This measure is the average fraction of each plagiarism case that is detected.

$$recall(S, R) = \frac{1}{|S|} \sum_{s \in S} \frac{\left|\bigcup_{r \in R}(s \sqcap r)\right|}{|s|} \tag{2}$$

Granularity This measure is the average number of detections outputted by the system for each actual detected plagiarized passage.

$$granularity(S, R) = \frac{1}{|S_R|} \sum_{s \in S_R} |R_s| \tag{3}$$

where S_R are the cases in S that are detected in R, and R_s are the detections in R that detect a given $s \in S$. The optimal and minimum value of the granularity is 1, which means that plagiarized sections are detected exactly once, and the worst value is $|R|$, which can be when all the detections are of the same $s \in S$ (which also means a poor recall for the other cases in S). This measure does not have an Information Retrieval equivalent, and it can somewhat measure the "usability" of the algorithm, in the sense that it is preferred for the final user to have one detection per actual case, even though the detection can be of just one character of the actual case, than many ones, that do not give more information than just one detection per case.

Plagdet This measure is a combination of the above measures, which allows comparisons between different methods.

$$plagdet(S, R) = \frac{F_\alpha}{log_2(1 + granularity(S, R))} \tag{4}$$

where F_α is the harmonic mean of the precision and recall defined before.

Finally, though the text alignment task was abandoned in PAN, there have been other developments in plagiarism detection outside it. In Gipp et al. (2014), the authors describe a mechanism for citation-based plagiarism detection, as opposed to the character-based detection used in most of the approaches at the PAN competition. In Abdi et al. (2015), the authors use WordNet, a lexical database where words are organized into sets of synonyms (Miller, 1995), to handle false positives that may arise from the change of meaning that can happen in sentences that have common words but are not paraphrasing. In Vani and Gupta (2017) the authors use a genetic algorithm to extract concepts from the document, in an attempt to handle obfuscated plagiarism. In Paul and Jamal (2015), the authors use Semantic Role Labelling (Gildea and Jurafsky, 2002), a technique that associates roles (like "subject" or "object") to terms in sentences, adding more information to the terms in the documents and allowing their method to handle paraphrasing. A comparison and categorization of many techniques for detecting plagiarism can be found in Sahi and Gupta (2016), and an state of the art review can be found in Meuschke and Gipp (2013), showing the maturity of the subject.

2.2. Large document collections

First insights on scalability awareness in plagiarism detection, though more focused on computer code, are given in Burrows et al. (2007), where an inverted index (Baeza-Yates et al., 1999) is used to compute similarities in a document collection in an all-versus-all mode. This data structure is used in many of the prior-to-2013 PAN iterations where the inverted index is used as a search engine, being queried with fragments of the suspicious documents to get the most likely sources. Another approach is to use an inverted index directly to find the plagiarized document pairs, without relying on query extraction from suspicious documents, as in Zhang et al. (2010).

Also, in Zhang et al. (2010), scalability issues are dealt with using the MapReduce framework (Dean and Ghemawat, 2008), which can be used to work with data that does not fit in memory by storing it on a disk and using multiple machines to process it. Along the same lines, this framework is used in Elsayed et al. (2008), Xu et al. (2011), Wu et al. (2011) and Wang et al. (2010), although they are more focused on scale than in the plagiarism detection algorithm.

2.3. Plagiarism detection systems

A state of the art for plagiarism detection systems can be found in Kakkonen and Mozgovoy (2010), where they also do an empirical evaluation of some real-world commercial systems. Before that, the idea of fully-fledged systems for plagiarism detection with an orientation towards the user, as opposed to just algorithms, can be traced back to Si et al. (1997), where user-oriented issues are discussed, like a mechanism for text extraction and the idea of creating custom databases of original documents. The idea of deploying a plagiarism detector as a distributed system is discussed in Monostori et al. (2000), showing that the approach can greatly improve the running time. The last version of the system proposed in this document was created in Velásquez et al. (2016), where architectural and technological issues are discussed in a system that not only does plagiarism detection, but also other text processing tasks, like intrinsic plagiarism detection (Oberreuter and Velásquez, 2013), quotation detection and document clustering.

Since plagiarism detection has a direct application, commercial engines have emerged, especially as web services. Here are some examples.

Turnitin (0000) One of the tools evaluated in Kakkonen and Mozgovoy (2010). A platform for education, which incorporates originality checking among its functionalities. This system uses its originality check as part of a suite of functionalities for writing better documents.

Grammarly (0000) A suite of tools for improving writing. One of them is the Plagiarism Checker, which searches text on the web.

iThenticate (0000) From the same company as Turnitin, this is an academia-oriented platform, which offers searching documents against the web and academic repositories, such as journals and magazines.

VeriGuide (0000) Another education-oriented platform which can support text in English and Chinese.

2.4. Contributions of this paper

In this paper we present the next iteration of Velásquez et al. (2016), where a suite of algorithms is presented for text analysis. However, pitching the system to the market has shown that the main focus has to be on external plagiarism detection, more than on other text processing tasks, and beyond detecting subtle plagiarism cases, simply detecting the "easy" ones. Also, no consideration is given to big document collections, while as described before, the size of the document collections is a relevant issue. The contribution of Velásquez et al. (2016) to this work is that, as our first attempt at a commercial system, it showed us the market-related problem mentioned before, and, through the people involved in that project, gave us insights about the general architecture of this system and the technologies to use in it.

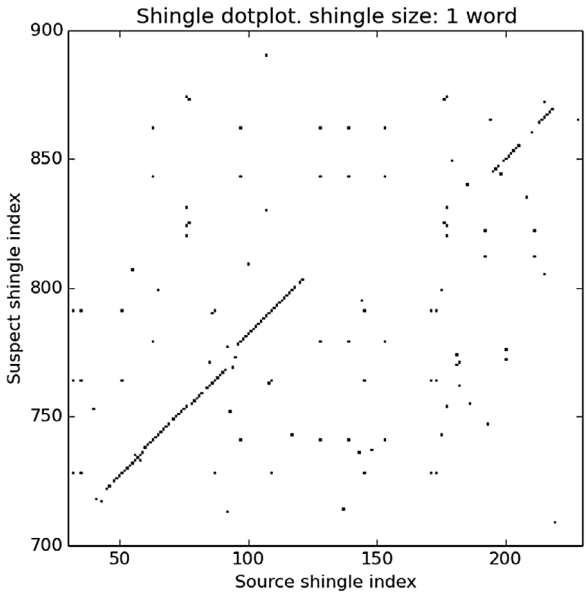
In this paper we present a fully-fledged plagiarism detection system for commercial use, we describe its architecture and most important, show the considerations we have when dealing with a large-scale deployment, that is, when dealing with big document collections and with the World Wide Web.

3. Proposed system

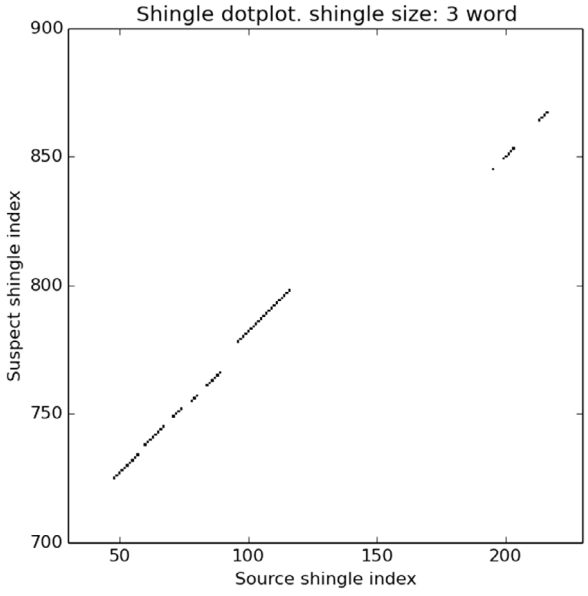
3.1. The core algorithm

The core algorithm for comparing two documents, used in all the functionalities of the system, is based on the analysis of the shingles of the document. The shingles (Broder, 1997) of a text, are the words yielded by a sliding window of a determined size over the words of the text. For instance, the shingles of size 3 of the text

```
this is the user interface
```



(a) Shingle size of 1 word



(b) Shingle size of 3 words

Fig. 1. Dotplot of two documents with reused text. The plots were shifted to make the lines appear centered for convenience.

are the list

```
    this is the
    is the user
the user interface
```

First, we strip the text of punctuation and stopwords, then with the shingles of the texts a dotplot is built, with the "suspect" and the "source" document for each axis. The dotplot (Helfman, 1996) is a visual technique to compare two sequences s and s' (in this case, the shingles of the source and the suspicious document). The axes of the dotplot are the indices in the sequences, and the point with coordinates (i, j) in the dotplot is marked if $s_i = s'_j$. Figs. 1a and 1b show dotplots of two documents with different shingle sizes.

Matching shingles between both documents are shown as dots in the dotplot. Text reuse happens when the shingle n of the suspect matches with the shingle m of the source, and so do suspect's $(n+1)$-th and source's

$(m+1)$th shingles, and so on, so plagiarized passages can be seen as continuous diagonal lines of matching shingles. Fig. 1 shows that small shingle sizes show more isolated coincidences, such as compound names like "Christopher Columbus", which can be expected to be reused in a group of documents about the Age of Discovery, for example, without making a plagiarism case, while introducing noise to the plot, making it more difficult to detect clear lines, whereas bigger shingle sizes generate a cleaner dotplot, but can cut some of the lines. More quantitatively, in Fig. 1a, there are 183 matching shingles, and 45.36% of them are in an empty neighborhood, whereas in Fig. 1b there are only 62 matching shingles, with only 1.61% of them without neighbors. As explained before, isolated matching shingles are of little use, so it is desired for a dotplot to have the least amount of them and having continuous lines instead when there is text reuse.

With the preprocessing steps described before, we no longer work with text and instead only work with the dotplot. Algorithm 1 shows the procedure for computing the matching shingles from a dotplot used in the system. This algorithm receives the list of dots of the dotplot, the maximum gap allowed between plagiarism cases, which defines how far two plagiarism cases have to be in order to be considered as just one case, and the minimum size for a plagiarism case to be considered. It is important to remark that this algorithm (with the preprocessing steps) is symmetric, that is, the results are the same if the inputs are swapped. Since we also remove stopwords in order to improve recall, the core algorithm is language-dependent, but the set of stopwords can be changed for other languages.

Algorithm 1 Matching documents

```
procedure GETMATCHES
Require:
  matchingIndices: List of pairs of integers
  maxGap: Maximum gap between clusters, integer
  minSize: Minimum cluster size
Ensure: clusters: List of lists of pairs of integers
    clusters ← ∅
    while matchingIndices ≠ ∅ do
        fmIndex ← pop first pair from matchingIndices
        sort matchingIndices according to the Chebyshev distance to fmIndex
        cluster = {fmIndex};
        while matchingIndices ≠ ∅ ∧ Chebyshev(matchingIndices.first, cluster.last) = 1 do
            Pop matchingIndices.first to cluster
            sort matchingIndices according to the Chebyshev distance to cluster.last
        end while
        add cluster to clusters;
    end while
    Merge cluster ∈ clusters closer than maxGap according to Chebyshev distance
    Remove cluster ∈ clusters smaller than minSize
    return clusters
end procedure
```

The Chebyshev distance between two matching shingle indices $a = (susp_i, src_j)$ and $b = (susp_k, src_l)$ is defined as the distance on the axis with the largest one, that is

$$Chebyshev(a,b) = max\{|susp_i - susp_k| - |src_k - src_l|\}. \quad (5)$$

With this metric we can compare how close are two matching shingles to determine if they belong to the same cluster and to determine if two clusters are close enough to be considered a single one (which leads to a single detection).

The maximum gap and the minimum cluster size parameters can be tuned by applying the algorithm to a dataset and evaluating the performance of the algorithm, with one of the performance measures described in Section 2.1. When tuning the algorithm we have seen that a small minimum cluster size leads to more detections, but also more spurious ones, giving more recall but less precision, whereas a big minimum cluster size can lead to less detections, because the minimum case length has to be bigger to be detected, giving more precision but less recall. Also, we have seen that a small maximum gap leads to fragmented detections (for instance, if a few words change in the plagiarized document, the algorithm yields two detections for just one plagiarism case), increasing the granularity, whereas a big maximum gap can lead to detect adjacent cases as one, with all the non-matching text between the detections, which decreases the granularity, but also decreases the precision of the algorithm.

3.2. Web analysis

The main functionality of the system is the web analysis where a document is analyzed to find reused text in common with documents on the internet. In Fig. 2 we can see the pipeline, the steps and modules involved in web analysis. First, text is extracted from the suspicious document. Then, inspired by Williams et al. (2014), the queries are generated by taking the list of nouns in the document and separating them in groups of some size. These queries that are passed to a search engine, which returns a list of links, each one with a snippet. In order to minimize the number of downloads, the snippet is used to discard results that are not likely to have matches. To do this, we take the words of the snippet and check the fraction of them that are in the suspicious document. If the fraction is below a threshold, the link is discarded. The threshold, in this case, was set to 0.1, meaning that links with snippets with less of the 1% of their words in the suspicious document are discarded. The remaining search results are sent to the document retriever, which retrieves the candidate sources for further analysis. We then have a set of potential source documents, which may or may not have matches with the suspicious document. The text of every source candidate is extracted and detailed comparison is made with the suspicious documents using algorithm 1 generating the final results.

The query generator module uses the Stanford part-of-speech tagger (Toutanova et al., 2003) to recognize nouns, making it dependent on the language, but, as with the core algorithm, the language model used by the tagger can be changed for other languages.

3.2.1. Issues with web analysis and proposed improvements

The web analysis task involves issuing requests to web pages, the same way web crawlers do (Olston and Najork, 2010). There are sites that are more likely to be reached by the system and, if requests are made without any control, the system can perform (unintentionally) a denial-of-service attack, which can end up with the system losing access to a potential source, and causing the web analyses to be incomplete. This can happen because there are sites that have lots of information, and are very likely to be used for text reuse, such as Wikipedia, and also because, in an educational environment, it is likely that similar documents (e.g. two assignments on the same subject) can lead to the same candidate source (regardless of actual text reuse). We use a politeness policy of throttling the requests per unit of time. To do that, the document retrieval module buffers every download on a per-domain basis, that is, when it receives a url to download, it puts it in a queue according to the url domain, then it downloads the pages from each domain one at a time with a delay between each download, so that the analysis does not overload a domain. Also, the document retriever module looks up the urls in a cache database, so that web pages are downloaded only once. This way, we deliver results as fast as possible, while assuring the quality of the web analyses. It is important to remark that here we assume the plagiarizable content on the web to be static, as we currently do not have any cache refreshing strategies implemented.

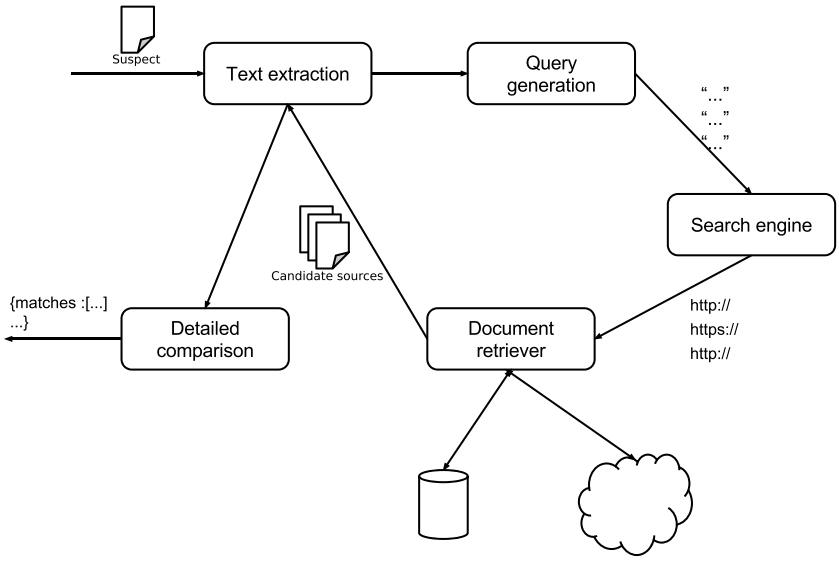


Fig. 2. Web analysis pipeline.

3.3. Custom repository analysis

The system can be used with a custom collection of source documents. Collections are defined by the user and suspicious documents can be compared with them, by using the same pipeline as described in 3.2, but instead of issuing queries to an online commercial search engine, it issues the queries to an internal search engine, and retrieves the documents from there, without downloading anything from the Internet. Of course, no considerations about the document downloads have to be made, since the storage is in a local network under our control.

3.4. Group analysis

When comparing a group of documents as all-versus-all (a group of student papers for the same homework, for instance), we could use the main method and build a matrix of document comparisons. But this is not the most efficient method, because the documents have to be preprocessed into shingles many times. Instead, we use a method based on (Zhang et al., 2010), in two phases, two MapReduce jobs, one for indexing the documents and one for analyzing them.

Algorithm 2 Indexing job

```
procedure MAP
Require:
  text: Text of the document, string
  id: Document identifier, integer
    for shingle ∈ GETSHINGLES(text, shingleSize) do
      emit(shingle.words, (id, shinglePos))
    end for
  end procedure

  procedure REDUCE
Require:
  shingle: String
  postings: List of the form (id, shinglePos, shingle.start, shingle.end)
    if postings.length ∈ [2, MAX_LENGTH] then
      emit(shingle, postings)
    end if
  end procedure
```

Algorithm 2 shows a standard index construction, which yields an index of the form $words : [(docId, shinglePos, shingle.start, shingle.end), \ldots]$, where $words$ are the words of the shingle, $docId$ is the document identifier, $shinglePos$ is the position of the shingle in the document, and $shingle.start$ and $shingle.end$ are the string indices of the shingle in the document, used later to indicate the final reused text spans. In order to make the index manageable, we discard in the reduce phase all shingles with only one entry, since they do not carry information about plagiarism, while not linking any pair of documents, and shingles whose entries have too many elements, since shingles that are used in too many documents are likely not to be plagiarism, but rather template text like headers and footers. The MAX_LENGTH parameter was set to 100, since bigger values made the computation infeasible for our infrastructure. Algorithm 3 shows the matching job, which uses the index built by the indexing job as input. The map procedure generates lists of the form $(doc_i, doc_j) : [(shingle_i, shingle_j), \ldots]$ which link two documents with the matching indices, which are then used with the algorithm 1 to get the matches and the final result. With these two jobs we can analyze a group of documents efficiently, without considerable redundancy[2] in obtaining the shingles, and in leveraging the computing

Algorithm 3 Matching job

```
  procedure MAP
Require:
  key: Words of the shingle, String
  posts: List of the form (id, shinglePos)
    for (post_i, post_j) ∈ posts × posts do
      emit((post_i.id, post_j.id), (post_i.shinglePos, post_j.shinglePos))
    end for
  end procedure

  procedure REDUCE
Require:
  (textId_i, textId_j): pair of document ids
  matchingIndices: List of the form (post_i.shinglePos, post_j.shinglePos)
    clusters ← GETMATCHES(matchingIndices, maxGap, minSize)
    for cluster ∈ clusters do
      emit((textId_i, textId_j), [(cluster.start, cluster.end)])
    end for
  end procedure
```

[2] In the index analysis reduce procedure there might be repeated shingles, but this can only happen when a section of a document is used more than once in another document.

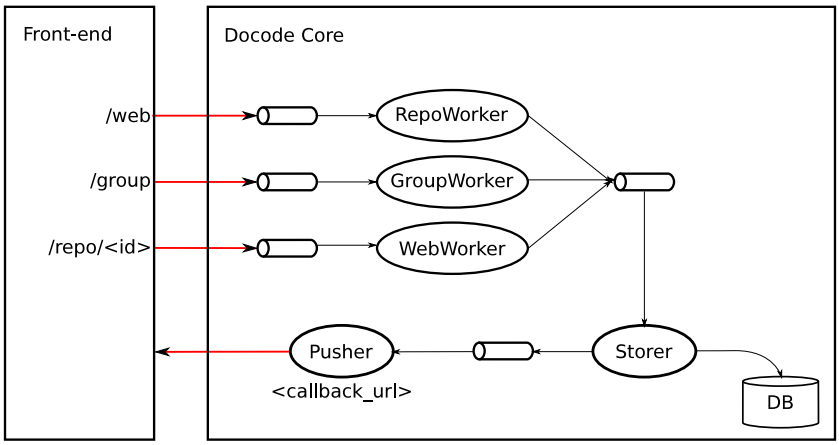


Fig. 3. Core high-level architecture.

power of a cluster of computers. It is important to remark that, since algorithm 1 is symmetric, the results of the jobs are the same as the ones we could obtain by building the document comparison matrix, even though we cannot control the ordering of the pair of documents being analyzed.

4. Architecture

The system is mainly composed of two loosely coupled subsystems, the core and the front-end, which communicate with each other through HTTP, with a JSON-based data exchange protocol. This makes it possible to change the front-end or make the core available to third-party applications.

4.1. Docode core

This is the core of the system, where the analyses are done and the functionalities of the system are implemented. In Fig. 3 the core architecture is shown. The core is developed as a Java Enterprise Application, and deployed to a Glassfish server (or any JEE compatible server for that matter), exposing an asynchronous HTTP interface, where the core receives requests from a client, with the required data to fulfill a request, and the core immediately returns an identifier for the process in case of errors. The functionalities described in Sections 3.2–3.4 are implemented in the worker nodes, `WebWorker`, `RepoWorker`, and `GroupWorker`, respectively (we could say Fig. 2 is inside the `WebWorker` node in Fig. 3). These nodes take the messages from a JMS queue the front-end has put the messages in, compute the results and send them to the `Storer` node, which puts the results in a database, and then sends them to the `Pusher`, which sends the results back to the client, using a callback URL. In case there is a problem sending the results to the client, the identifier can be used to retrieve the result from the database.

4.1.1. Technologies used in the system

There are many submodules in the system that have an open source, or at least accessible, implementation, making it unnecessary for us to implement them. For the text extraction, for all nodes, we use the Tika library (Tika, 0000). The Tika library combines many other text extraction libraries for different filetypes, and has an automatic type detection functionality, allowing, with an unified API, for text extraction from different filetypes, such as Adobe's PDF and Microsoft Word's DOC and PPT formats. Since we want to deploy our system commercially, it becomes necessary to make it open to the most popular document formats.

For the `WebWorker`, we use Google's Custom Search API (goo, 0000) as a search engine. This is a service provided by Google that allows to search the web programmatically, the same way a user can search the web through their website. We need this technology because it is too much of an entry barrier having to implement a search engine as good as Google's to deploy our system. It is important to remark that this is a paid online service, so its use involves an operational cost.

For the `RepoWorker`, the documents are stored in a Solr database. Solr (0000) is a database that allows for full-text search, something like a search engine in a group of files. Solr, in this system, works both as a storage and a search engine for the repositories, so the repositories can be searched and the document texts can be retrieved from there.

For the `GroupWorker`, we run the MapReduce jobs with Hadoop. Apache Hadoop (0000) is an open source implementation of the MapReduce framework, which allows for the processing of large datasets in clusters of computers, offering a data-redundant file system and the machinery for distributing jobs in a cluster of computers. In a Hadoop cluster there is usually a master machine, which distributes the jobs and the data to process, and many slave machines, which do the actual computation in the cluster.

4.2. Docode front-end

This module works as the user interface. Here the analyses are shown to the end user and document repositories are managed. This module also manages the user profiles. Currently we have two front-ends, one for use by individuals, which offers web analyses, and one for use by organizations (which offer the services themselves to their own people), which offers web analyses, group comparisons and repository analyses.

4.2.1. Single user front-end

Figs. 4, 5a and 5b show the main views for the front-end. Fig. 4 shows the dashboard, where a user can upload documents to the system, get them analyzed and get the results. Since the back-end is asynchronous, the front-end has to keep track of the analyses done previously, so that the user can check the results. Fig. 5 show the views for a plagiarism case. Fig. 5a shows the global report for one document. This report shows for each document the global plagiarism ratio, that is, the fraction of the words of the document that are involved in plagiarism cases, and, for each source, the number of matches and the local plagiarism ratio, that is, the fraction of the document involved in the plagiarism case with that particular source. It is important to remark that the local plagiarism rates do not necessarily add up to the global plagiarism ratio, because a passage from a suspect can be plagiarized from more than one source. A problem that we found with the previous version of the system was that the measurements for plagiarism were deemed too complicated for helping in the decision making process. In this iteration, we took care to make the displayed rates as simple as possible, by displaying just one

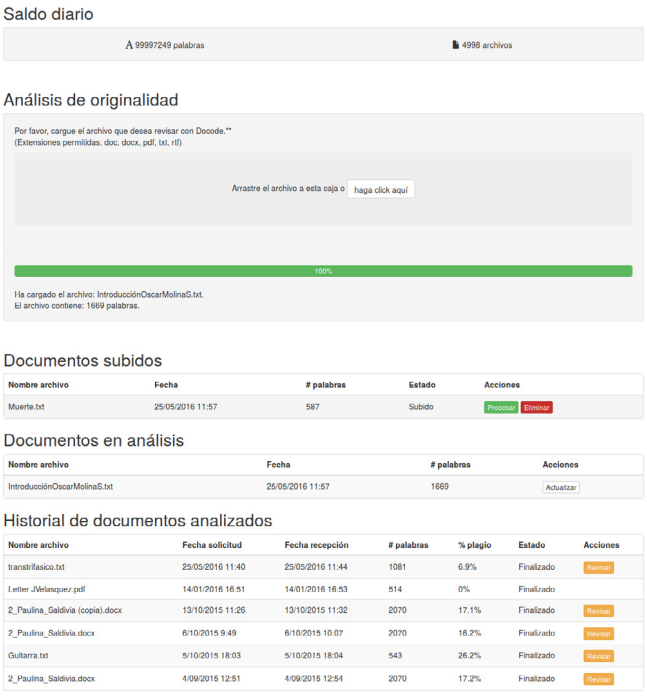


Fig. 4. Front-end dashboard.

number, the *plagiarism index*.[3] In Fig. 5a, the "global" plagiarism index is the fraction of the words of the suspicious document that are in any of the found sources, whereas in Fig. 5b the "local" plagiarism index is the fraction of the words of the suspicious document that are in one specific source. The detections in different sources may overlap in the suspicious document, so the local plagiarism indices do not necessarily add up to the global plagiarism index. Also, since this system is ultimately a decision-support system, it is up to the user to decide if the text reuse found is really plagiarism, with all the implications this decision can have (such as legal ones). Finally, Fig. 5b shows a detailed report for one source. As with the previous iteration, it shows the suspect and the source in a plain-text representation and highlights the plagiarized passages of text, along with the local plagiarism index for the source.

4.2.2. Front-end for organizations

The single-user front-end described in Section 4.2.1 is the "lite" version of the system, offered to the public. The front-end for organizations is the "full" version of the system. In the full version an administrator can manage privileges for users and, most important, they can use the other functionalities of the system, namely the repository analysis and the group analysis. Since, as said in Section 3.3, the two workflows (web and repository analyses) are quite similar, the views are similar to Fig. 5. Thus, we show that, with this modular architecture, we can have a core which does the real job, and more than one view to show it in, allowing us to eventually make other workflows for new types of users (see Fig. 6).

5. Experiments

5.1. The core algorithm

We tested the algorithm 1 with an internal corpus of documents in Spanish and the non-obfuscated portion of the PAN-PC-13 (Potthast et al., 2013) test corpus, in English. The corpus for Spanish is very small,

[3] Translated to Spanish to *índice de plagio* in Fig. 5.

(a) Global report for a document.

(b) Detailed report for a source.

Fig. 5. Report for a repository analysis.

Table 1
Performance of the document matching algorithm with an internal corpus.

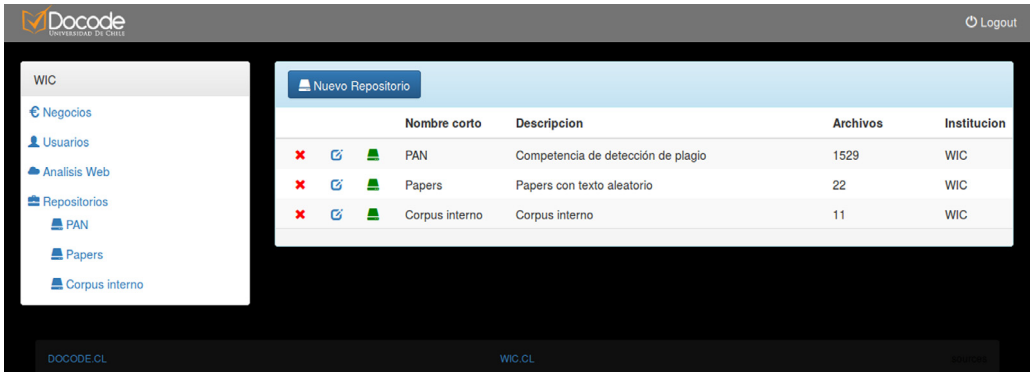
Precision	Recall	Granularity	Plagdet score
0.99	0.90	1.00	0.94

with 20 plagiarism cases, but has exactly what we need to evaluate for our commercial deployment, whereas the PAN-PC-13 corpus is a well-established corpus for plagiarism detection evaluation, with 1000 plagiarism cases (the subset we use). The results are shown in Tables 1 and 2. The performance measures used are the ones defined in Potthast et al. (2010b).

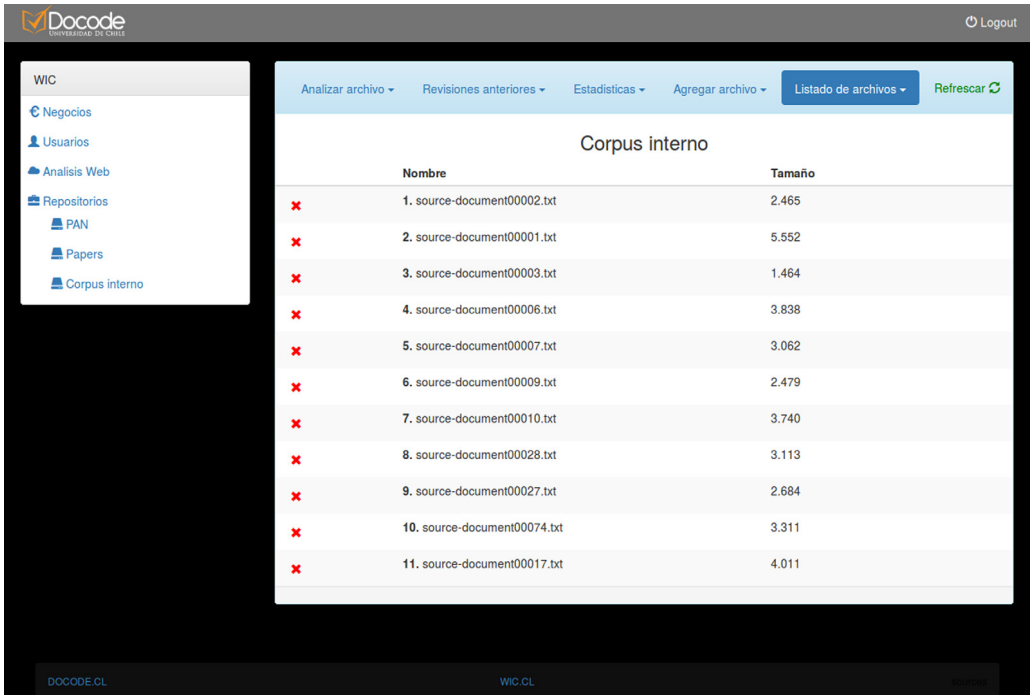
5.2. Group analysis

For these tests, we run a cluster of 5 Amazon EC2 machines, each one with 4 GiB of RAM. 2 CPU cores, with 1 master machine, and 4 slave machines. The corpus used for these experiments was made of the PAN-PC-13 corpus (4995 documents), a corpus we know has plagiarism cases in it, as well as articles from the English version of Wikipedia. First, we tested the impact of adding more nodes to the cluster, starting with only 2 slave nodes up to 4 slave nodes (making full use of the cluster), with a 40168-document collection. The results are shown in Fig. 7.

In order to show the performance of the jobs in the cluster, we run the algorithms with a growing collection of documents, starting with



(a) Global report for a document.



(b) Detailed report for a source.

Fig. 6. Report for a repository analysis.

Table 2
Performance of the document matching algorithm with the non-obfuscated part of PAN-PC-13, comparing each measure with the best and the worst of Potthast et al. (2013).

	Precision	Recall	Granularity	Plagdet score
Best	0.985 (Jayapal, 2012)	1.00 (Oberreuter et al., 2014)	1 [a]	0.94 (Oberreuter et al., 2014)
Docode 5	0.881	0.99	1.03	0.91
Baseline				0.88

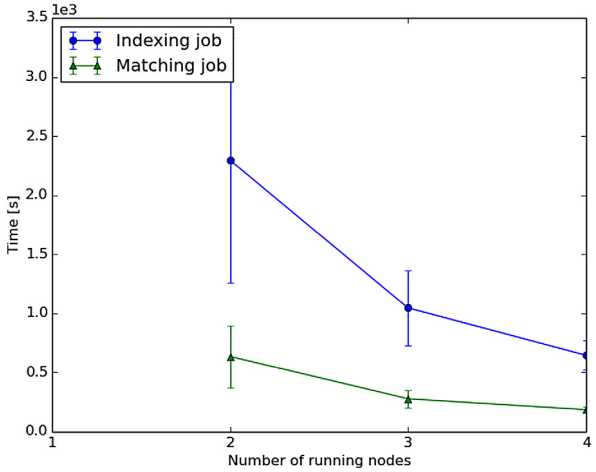
[a] 15 of the 19 submissions tested in Potthast et al. (2013) reached this granularity, so, for the sake of space, we omit them.

a base collection made of the PAN-PC-13 corpus (4995 documents), and then we add documents from the English Wikipedia, up to 90 000 documents. That is, we measured the time for our algorithm for group analysis 10 times on the PAN-PC-13 corpus, and then added about 10 000 documents from Wikipedia to the group, measured the time to process the new 10 times, and so on, up to the 90 000 documents. Also, we measured how the index expanded, that is, how many different shingles were registered in the index, and how many plagiarism cases were detected as we analyzed more and more documents. The results are shown in Fig. 8.

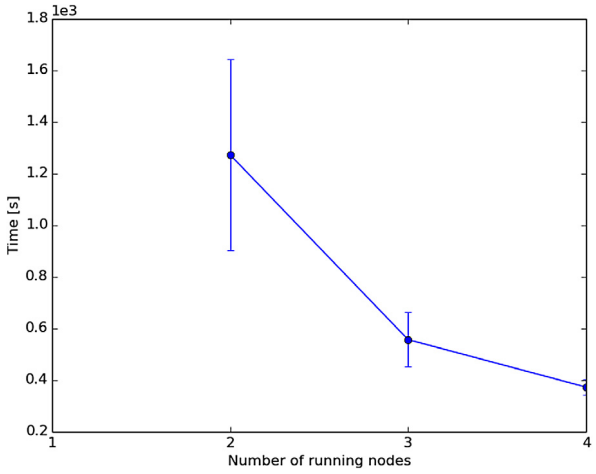
5.3. Discussion

In Tables 1 and 2 we can see that the performance of the core algorithm (the one the other functionalities of the system are based on) is over 90%, which makes it fit for our commercial purpose, while also validating it with a standard dataset. We can expect the performance of the system to be slightly higher in our dataset than in a standard dataset, because the main focus of the system is satisfying our commercial requirements for plagiarism, expressed in the internal dataset.

Fig. 7 we see that the algorithm can leverage the computing power of a cluster, giving a result in less time with more running nodes. In

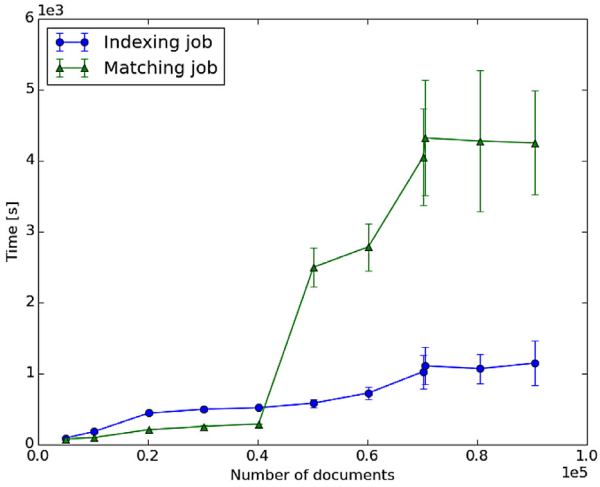


(a) Elapsed times of the matching and indexing jobs vs. number of running nodes.

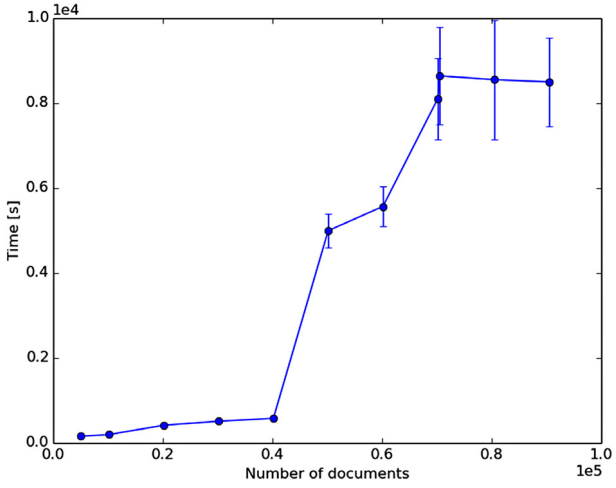


(b) Total elapsed times for the group analysis jobs vs. number of running nodes.

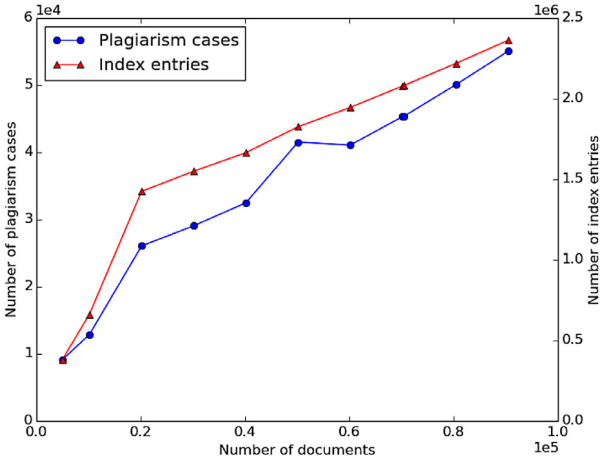
Fig. 7. Impact of number of nodes in the running time of the group analysis algorithms.



(a) Elapsed times of the matching and indexing jobs vs. number of documents.



(b) Total elapsed times for the group analysis jobs vs. number of documents.



(c) Number of entries in the index and number of plagiarism cases vs. number of documents.

Fig. 8. Impact of number of documents in the running time of the group analysis algorithms.

Fig. 8a we see that in collections with fewer than 40,000 documents, the majority of the running time of the analysis is in the indexing job, whereas with bigger collections the majority of the cost is given by the matching job. Looking to the points in Fig. 8c we can see that, with the collection used in the experiments, the batch of documents added between the 40,000 and 50,000 document increments, lead to more plagiarism cases than the other batches we add. This shows that the amount of work the matching job does depends on the number of plagiarism cases in the collection.

A problem we found was that with document collections that are too big there are some plagiarism cases that are ignored. This can be caused by the restriction imposed on the length of the index entries (the MAX_LENGTH parameter in Algorithm 2). When more and more documents are added to the collection, some entries in the index exceed the MAX_LENGTH parameter, getting them removed from it and causing some plagiarism cases to be ignored. Finally, the larger standard deviations in the bigger collection analyses in Fig. 8b suggest that the cluster can be tuned to get better performance times.

6. Future work

Though, as shown, the overall architecture of the system is robust enough to do a commercial deployment, most of the modules of the system could be improved.

In the text extraction module, we are using the Tika library in all functionalities of the system. This parser has some problems dealing with PDF files and it does not extract structure from the files, thereby introducing noise to the matching algorithm. Improving the parser

to recognize structure in documents would allow the core algorithm and the query generator to get better results, improving the overall performance of the system.

The front-end presents its results in an effective way, but this could be improved by showing the results in the documents themselves instead of in a plain text representation, thus improving the usability to the system.

The strategy for query extraction, though is good enough to find source documents when there is plagiarism, issues too many useless queries to the search engine, so better strategies could be analyzed, in the sense that fewer queries could cover more text of the document. Also, we need to study the impact of issuing fewer queries (with any parameter-dependent strategy) in the recall of the system, in order to find the minimum number of queries required to find a certain recall. This way we could reach an acceptable compromise between the number of queries per document (which become an operational cost) and the recall of the system. This way we could start to reduce the operating costs of the system.

Finally, the main problem with the system currently is that it is dependent on a commercial search engine and the web in general, which involves, as said before, operational costs and also ties our system to the stability of the network, so if our internet connection becomes unstable, our service quality becomes unstable. With this in mind, future efforts have to be made to achieve independence from the web content. That can be done in two phases; first, we would have to crawl web pages in order to have them before they are needed for analysis, thus making the system more reliable and independent from the original content providers. Finally, we would have to build a search engine with the crawled web pages, making the system independent of the search engine, and considerably reducing the operating costs of the system.

7. Conclusions

In this paper we have presented Docode 5, a system for plagiarism detection, and we have shown the problems when dealing with big document collections and when dealing with the World Wide Web. We have shown that we can take an algorithm for two-document plagiarism detection and adapt it for doing all-versus-all document comparisons, and also that we can make a system for plagiarism detection on the web that can take into account its own impact on it, maintaining the quality of the performed analyses. We have shown our system from end to end, and all the components integrated for it, showing how we can use our components, along with other external implementations, to make a fully-fledged plagiarism detection system.

Finally, it is important to remark, as said before, that this is a decision support system, and, no matter how good the algorithms are, or how good the performance of the system is, this system can only be used as a tool, and the user is the one who has the final decision on a plagiarism case.

Acknowledgments

This work was supported partially by the Millennium Institute on Complex Engineering Systems (ICM: P-05-004-F, CONICYT: FBO16).

References

Apache Solr -, URL http://lucene.apache.org/solr/ (accessed: 13.06.17).
Apache Tika - Apache Tika URL https://tika.apache.org/ (accessed: 13.06.17).
Abdi, A., Idris, N., Alguliyev, R.M., Aliguliyev, R.M., 2015. PDLK: Plagiarism detection using linguistic knowledge. Expert Syst. Appl. 42 (22), 8936–8946.
Baeza-Yates, R., Ribeiro-Neto, B., et al., 1999. Modern information retrieval, Vol. 463. ACM press New York.
Broder, A.Z., 1997. On the resemblance and containment of documents. In: Compression and Complexity of Sequences 1997. Proceedings. IEEE, pp. 21–29.
Burrows, S., Tahaghoghi, S.M., Zobel, J., 2007. Efficient plagiarism detection for large code repositories. Softw. Pract. Exp. 37 (2), 151–176.
Clough, P., et al. 2003. Old and new challenges in automatic plagiarism detection, in: National Plagiarism Advisory Service, Citeseer, 2003; URL http://ir.shef.ac.uk/cloughie/index.html.
Custom Search — Google Developers URL https://developers.google.com/custom-search/ (accessed: 13.06.17).
Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51 (1), 107–113.
Elsayed, T., Lin, J., Oard, D.W., 2008. Pairwise document similarity in large collections with MapReduce. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers. Association for Computational Linguistics, pp. 265–268.
Gildea, D., Jurafsky, D., 2002. Automatic labeling of semantic roles. Comput. Linguist. 28 (3), 245–288.
Grammarly - plagiarism checker URL https://www.grammarly.com/plagiarism-checker (accessed: 13.06.17).
Gipp, B., Meuschke, N., Breitinger, C., 2014. Citation-based plagiarism detection: Practicability on a large-scale scientific corpus. J. Assoc. Informat. Sci. Technol. 65 (8), 1527–1540.
Hagen, M., Potthast, M., Stein, B., Source Retrieval for Plagiarism Detection from Large Web Corpora: Recent Approaches.
Hannabuss, S., 2001. Contested texts: issues of plagiarism. Lib. Manag. 22 (6/7), 311–318.
Helfman, J., 1996. Dotplot patterns: a literal look at pattern languages. TAPOS 2 (1), 31–41.
iThenticate URL http://www.ithenticate.com/ (accessed: 13.06.17).
Jayapal, A., 2012. Similarity overlap metric and greedy string tiling at pan 2012: Plagiarism detection, in: CLEF (Online Working Notes/Labs/Workshop).
Kakkonen, T., Mozgovoy, M., 2010. Hermetic and web plagiarism detection systems for student essays—an evaluation of the state-of-the-art. J. Educ. Comput. Res. 42 (2), 135–159.
Meuschke, N., Gipp, B., 2013. State-of-the-art in detecting academic plagiarism. Int. J. Educ. Integr. 9 (1).
Miller, G.A., 1995. WordNet: a lexical database for English. Commun. ACM 38 (11), 39–41.
Molina, F., Velásquez, J.D., Rıos, S., Calfucoy, P.A., Cociña, M., 2011. El fenómeno del plagio en documentos digitales: un análisis de la situación actual en el sistema educacional chileno. Rev. Ing. Sistemas XXV.
Monostori, K., Zaslavsky, A., Schmidt, H., 2000. Document overlap detection system for distributed digital libraries. In: Proceedings of the Fifth ACM Conference on Digital Libraries. ACM, pp. 226–227.
Oberreuter, G., Carrillo-Cisneros, D., Scherson, I.D., Velásquez, J.D., 2014. Submission to the 4th International Competition on Plagiarism Detection, Notebook Papers of PAN CLEF.
Oberreuter, G., Velásquez, J.D., 2013. Text mining applied to plagiarism detection: The use of words for detecting deviations in the writing style. Expert Syst. Appl. 40 (9), 3756–3763.
Olston, C., Najork, M., 2010. Web crawling. Found. Trends Inf. Retr. 4 (3), 175–246.
Paul, M., Jamal, S., 2015. An improved SRL based plagiarism detection technique using sentence ranking. Procedia Comput. Sci. 46, 223–230.
Potthast, M., Barrón-cedeño, A., Eiselt, A., Stein, B., Rosso, P., 2010a. Overview of the 2nd international competition on plagiarism detection, in: In Proceedings of the SEPLN'10 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse.
Potthast, M., Gollub, T., Hagen, M., Graßegger, J., Kiesel, J., Michel, M., Oberländer, A., Tippmann, M., Barrón-Cedeño, A., Gupta, P., Rosso, P., Stein, B., 2012. Overview of the 4th International Competition on Plagiarism Detection, in: P. Forner, J. Karlgren, C. Womser-Hacker, (Eds.), Working Notes Papers of the CLEF 2012 Evaluation Labs ISSN: 2038-4963, ISBN: 978-88-904810-3-1 URL http://www.clef-initiative.eu/publication/working-notes.
Potthast, M., Gollub, T., Hagen, M., Tippmann, M., Kiesel, J., Rosso, P., Stamatatos, E., Stein, B., 2013. Overview of the 5th International Competition on Plagiarism Detection, in: P. Forner, R. Navigli, D. Tufis, (Eds.), Working Notes Papers of the CLEF 2013 Evaluation Labs, ISSN:2038-4963, ISBN: 978-88-904810-3-1 URL http://www.clef-initiative.eu/publication/working-notes.
Potthast, M., Hagen, M., Beyer, A., Busse, M., Tippmann, M., Rosso, P., Stein, B., 2014. Overview of the 6th International Competition on Plagiarism Detection, in: L. Cappellato, N. Ferro, M. Halvey, W. Kraaij, (Eds.), Working Notes Papers of the CLEF 2014 Evaluation Labs, CEUR Workshop Proceedings, CLEF and CEUR-WS.org. ISSN: 1613-0073 URL http://www.clef-initiative.eu/publication/working-notes.
Potthast, M., Stein, B., Barrón-Cedeño, A., Rosso, P., 2010b. An evaluation framework for plagiarism detection, in: Proceedings of the 23rd International Conference on Computational Linguistics: Posters, Association for Computational Linguistics, pp. 997–1005.
Potthast, M., Stein, B., Eiselt, A., universität Weimar, B., Barrón-cedeño, A., Rosso, P., 2009. Overview of the 1st International competition on plagielarism detection, in: SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse PAN 09, CEUR-WS. Org, pp. 1–9.
Sahi, M., Gupta, V., 2016. Efficiency comparison of various plagiarism detection techniques. In: International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT). IEEE, pp. 2974–2978.

Si, A., Leong, H.V., Lau, R.W., 1997. Check: a document plagiarism detection system. In: Proceedings of the 1997 ACM Symposium on Applied Computing. ACM, pp. 70–77.

Stein, B., Barrón Cedeño, L.A., Eiselt, A., Potthast, M., Rosso, P., 2011. Overview of the 3rd International Competition on Plagiarism Detection, in: CEUR Workshop Proceedings, CEUR Workshop Proceedings.

Toutanova, K., Klein, D., Manning, C.D., Singer, Y., 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1. Association for Computational Linguistics, pp. 173–180.

Turnitin - technology to improve student writing, URL http://turnitin.com/ (accessed: 13.06.17).

Vani, K., Gupta, D., 2017. Detection of idea plagiarism using syntax–Semantic concept extractions with genetic algorithm. Expert Syst. Appl. 73, 11–26.

Velásquez, J.D., Covacevich, Y., Molina, F., Marrese-Taylor, E., Rodríguez, C., Bravo-Marquez, F., 2016. DOCODE 3.0 (DOcument COpy DEtector): A system for plagiarism detection by applying an information fusion process from multiple documental data sources. Inf. Fusion 27, 64–75.

VeriGuide - Plagiarism Detection and Document Analysis URL http://veriguide1.cse.cuhk.edu.hk/portal/plagiarism_detection/index.jsp (accessed: 13.06.17).

Wang, C., Wang, J., Lin, X., Wang, W., Wang, H., Li, H., Tian, W., Xu, J., Li, R., 2010. MapDupReducer: detecting near duplicates over massive datasets. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. ACM, pp. 1119–1122.

Welcome to Apache ™ Hadoop® URL http://hadoop.apache.org/ (accessed: 13.06.17).

Williams, K., Chen, H.-H., Giles, C., 2014. Supervised Ranking for Plagiarism Source Retrieval—Notebook for PAN at CLEF 2014, in: L. Cappellato, N. Ferro, M. Halvey, W. Kraaij, (Eds.), CLEF 2014 Evaluation Labs and Workshop — Working Notes Papers, 15-18 September, Sheffield, UK, CEUR-WS.org ISSN: 1613-0073.

Wu, Y., Zhang, Q., Huang, X., 2011. Efficient near-duplicate detection for Q&A forum, in: IJCNLP pp. 1001–1009.

Xu, F., Zhu, Q., Li, P., 2011. Detecting text similarity over chinese research papers using mapreduce. In: Software Engineering, Artificial Intelligence, 2011 12th ACIS International Conference on Networking and Parallel/Distributed Computing (SNPD). IEEE, pp. 197–202.

Zhang, Q., Zhang, Y., Yu, H., Huang, X., 2010. Efficient partial-duplicate detection based on sequence matching. In: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, pp. 675–682.