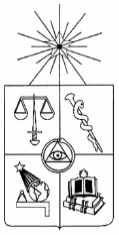

UNIVERSIDAD DE CHILE
DEPARTAMENTO DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE UNA TÉCNICA PARA LA DETECCIÓN DE PLAGIO EN DOCUMENTOS DIGITALES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INDUSTRIAL

GABRIEL IGNACIO LEÓN OBERREUTER GALLARDO

PROFESOR GUÍA:
JUAN D. VELÁSQUEZ SILVA

MIEMBROS DE LA COMISIÓN:
GASTÓN L´HUILLIER CHAPARRO
SEBASTIÁN RÍOS PÉREZ

SANTIAGO DE CHILE
2010

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INDUSTRIAL
GABRIEL IGNACIO LEÓN
OBERREUTER GALLARDO
FECHA: 25/10/2010
PROF. GUIA: SR. JUAN D.
VELÁSQUEZ SILVA

DESIGN AND IMPLEMENTATION OF A METHOD FOR PLAGIARISM DETECTION IN DIGITAL DOCUMENTS

With plagiarism one incur in an ethic fault as the original author gets no credit. In the case of written documents, one can commit this action including fragments without the proper citation, using the same ideas or copying the entire text.

Now, with the information technologies, such as Internet, a vast amount of easy to access information exist, therefore the plagiarism becomes a more viable and tempting option to students: to copy and paste a work one has to do for educational and investigational purposes becomes more recurrent.

The detection of said cases of plagiarism is complex, because of the infinite possible sources available. Then, automated plagiarism detection tools designed for lots of documents becomes more important. These tools are based in common pattern detection, information retrieval techniques and in the information theory.

One possible solution is the automated detection of verbatim plagiarism. Although other types of plagiarism exist, such as semantic plagiarism, as a premise we consider that textual copy represents a large proportion of the problem, and automated detection is possible and reliable.

The main objective of this thesis is to develop a method for automated plagiarism detection and implement it on a prototype. This method should include state of the art technologies and innovative tools to achieve its goal. Also, the system is going to be evaluated in an international workshop and competition in order to determine its effectiveness and to test its characteristics.

In this work we propose the design and an implementation of a method for automated detection of verbatim plagiarism. This method will then be used by DOCODE, project in which we develop a commercial product to be sold, which success depends in the efficacy and efficiency of the method here introduced.

The results indicates that an exhaustive search within a pair of documents can achieve the best results overall. But this exhaustive comparison has it's disadvantages; it is more expensive in terms of computation resources. The proposed method achieved acceptable results; it's F-measure is 0.8 in the dataset used, compared to 0.9 in the case of the exhaustive comparison, but it works requiring considerably less running time (25 percent the time the exhaustive approach required).

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INDUSTRIAL
GABRIEL IGNACIO LEÓN
OBERREUTER GALLARDO
FECHA: 25/10/2010
PROF. GUIA: SR. JUAN D.
VELÁSQUEZ SILVA

DISEÑO E IMPLEMENTACIÓN DE UNA TÉCNICA PARA LA DETECCIÓN DE PLAGIO EN DOCUMENTOS DIGITALES

Copiar o plagiar es cometer una falta ética al restar crédito al autor del trabajo original. En el caso de documentos escritos, se puede incurrir en copia mediante la inclusión de fragmentos sin citar la fuente original, plagiando ideas o mediante la copia completa del texto.

Con el uso de tecnologías de información, como Internet, existe una gran cantidad de información de fácil acceso, por lo que el plagio es una opción de gran tentación para los estudiantes: el acto de copiar y pegar durante la realización de informes o trabajos en el ámbito educativo y de investigación es un tema cada vez más recurrente.

La detección de dichos casos de copia es compleja, debido a la infinidad de fuentes disponibles. Debido a esto, herramientas de detección automática de plagio, diseñadas para grandes volúmenes de documentos, cobran mayor importancia. Estas herramientas se basan en la detección de patrones en común, en diferentes técnicas de recuperación de información y en la teoría de la información.

Una posible solución es la detección automatizada de copia textual. Como hipótesis, se postula que si bien existen otros tipos de copia, por ejemplo la copia semántica, la copia textual representa una porción importante de lo plagiado, y su detección automatizada es posible y precisa.

El objetivo principal de esta tesis es el desarrollar un método para la detección automatizada de plagio en documentos digitales, e implementar un prototipo para comprobar su efectividad. Este método debiese contar con tecnologías y conceptos de última generación para cumplir con su objetivo. Además, el sistema será evaluado mediante la participación en un taller y competencia internacional en detección de plagio.

En este trabajo, se plantea el diseño e implementación de un método de detección automático de plagio textual, basado en el lenguaje de programación java. Este método se acoplará posteriormente al sistema DOCODE (de ahora en adelante arquitectura DOCODE) proyecto en el cual se desarrolla un producto a comercializar, cuyo éxito de ventas depende, en parte, de la eficacia y rendimiento de la estrategia de detección aquí propuesta.

Los resultados indican que una búsqueda exhaustiva de plagio en un par de documentos puede obtener los mejores resultados en comparación a otros métodos. Pero este método tiene una desventaja; requiere el mayor tiempo computacional de todos los algoritmos probados. El método propuesto utiliza una aproximación y obtiene resultados aceptables; su F-measure es de 0.8 sobre los datos utilizados, comparado con 0.9 que el algoritmo exhaustivo obtiene, pero requiere considerablemente menos recursos computacionales (25 por ciento del tiempo total utilizado por el algoritmo exhaustivo).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Plagiarism exists since the born of mankind and basically consists in to take other's work and label it as one's own. In the case of text plagiarism, the copy is defined as the action of literally copy and paste someone else work without the proper citation [15].

Particularly, one can consider three cases of plagiarism: verbatim copy, to paraphrase and ideas copy, and self copy. In the first place, verbatim copy is when someone utilizes the text as is and provides no citation. To paraphrase is to use the idea changing the words. And finally self copy is to use self's work without citation.

Because there is a vast amount of easily to access information, thanks to information technologies such as Internet, the copy and paste phenomena has become more popular and easy to incur into. A few years before, the only possibility to detect such cases was to manually examine each document, a generally slow process. In this context, the entity Departamento de Ingeniería Civil Industrial of Universidad de Chile has postulated to a project, named Document Copy Detector (DOCODE from now on) and whose architecture is shown on Figure 1.1, to Fondo de Fomento de Desarrollo Científico y Tecnológico (FONDEF from now on). This project objectives are to design and to develop a system, capable of helping with the copy detection task. This system should use information technologies to do automated detection of textual plagiarism.

In Figure 1.1 a diagram of the system design is shown. One can see the different components; the databases, the algorithms for plagiarism detection, the "metasearch" engine, the interface, and others.

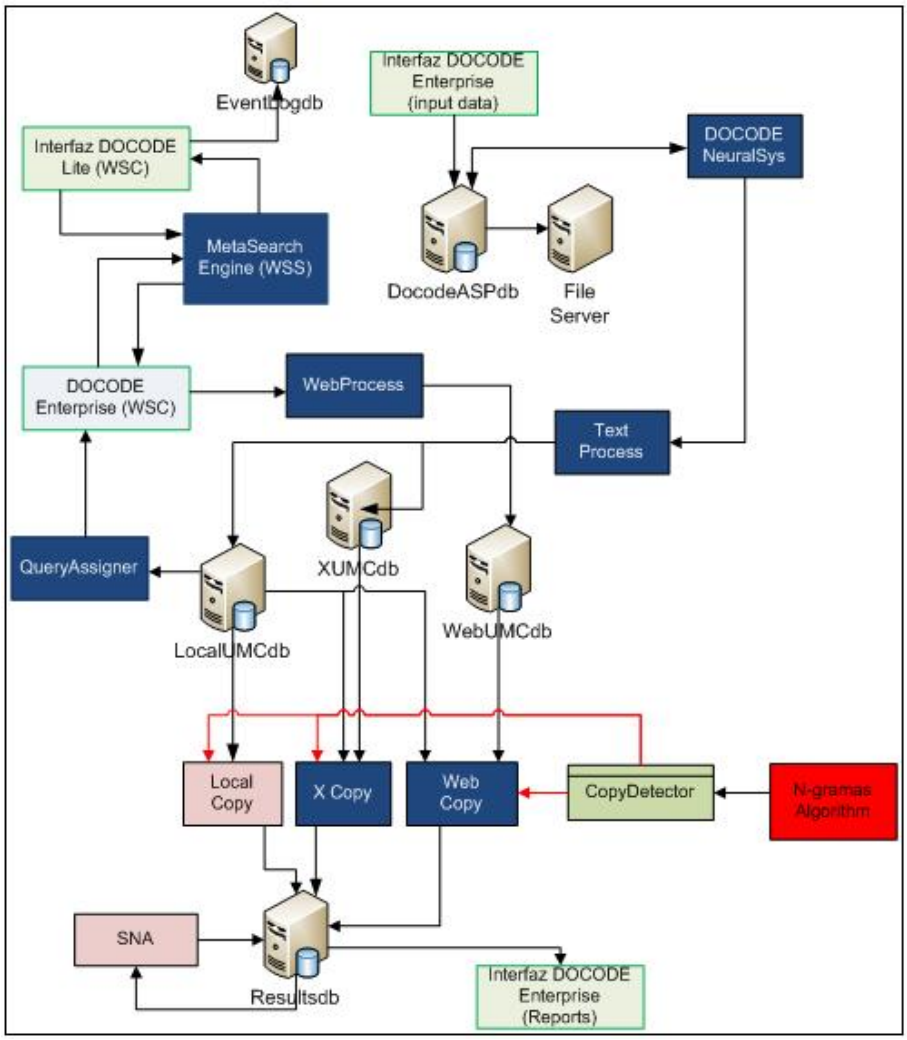

Figure 1.1: Architecture of DOCODE system.

1.1 Motivation

Given the large volume of documents and information sources that exist today, originality examination and plagiarism detection are complex tasks. Without doubt, this is an important issue for educational purposes, because plagiarism could affect the learning process of a student [26], and teachers often don't have the necessary time for exhaustive revision.

In [13] the authors conclude that "looking at the extend of the problem, it is quite obvious that academia requires tools to automate and enhance plagiarism detection". In accordance with [12] "plagiarism detection tools are programs that compare document with possible sources in order to identify similarity and so discover submissions that might be plagiarized".

Therefore, this work proposes a methodology for automated textual plagiarism detection, considering lot's of documents. This methodology is going to be used for the implementation of the plagiarism detection algorithm to be used by the DOCODE system, FONDEF project D08I-1015. This component will detect verbatim copy in document that come in majority from educational institutions. For example one kind of documents possible is student's homework.

DOCODE will be afterwards sold as a system able to support the plagiarism detection task.

1.2 Objectives

1.2.1 General Objective

To design and implement a technique for automated textual plagiarism detection considering lot's of documents, in the context of the FONDEF D08I-1015 project.

1.2.2 Specific Objectives

1. To document the state of the art in automated plagiarism detection in digital documents.
2. To design and to implement a methodology to extract, clean and process digital documents.
3. To design and to implement a technique for automated textual plagiarism detection.
4. To evaluate the developed algorithm efficiency and efficacy.
5. To integrate the methodology to DOCODE system.

1.3 Methodology

The methodology used to the development of this thesis, is structured in the following steps:

1. To define the problem to be solved.

   In this point general information is given and then the problem is defined.

2. To document the state of the art in the plagiarism detection task.

   The state of the art will be documented using publications and books as reference, to be able to build a Conceptual Framework chapter. In this chapter automated plagiarism detection will be discussed and plagiarism search space concept will be introduced. Specific objective number 1 is to be achieved.

3. To use extraction, cleaning and processing techniques in digital documents.

   Different techniques for documents preprocessing will be used, as recovering information within the documents, remove the stopwords and remove non alphanumeric characters. This process can be useful for large numbers of documents, for example the corpus of International Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection (PAN), which contains thousands of texts of different sizes [30]. Specific objective number 2 is to be achieved.

4. Design and implement a strategy for automated detection of plagiarism.

   An automated detection strategy based on word n-grams will be designed. Afterwards an implementation in java programming language will be implemented. Specific objective number 3 is to be achieved.

5. Continuous strategy evaluation and updates according to experiments results.

   Using iterative development ensures improving reliability and efficiency of the strategy. Specific objective number 4 is to be achieved.

6. Integration with DOCODE system.

   At this point the efforts will be oriented to integrate the methodology in DOCODE. A communication protocol will be established in order for the implementation to communicate with other components of the system. Specific objective number 5 is to be achieved.

Using this methodology and by achieving each specific objective, the main objective is to be completed.

1.4 Thesis Scope

The scope of this work is to develop a technique and a prototype of a external plagiarism detection system. The system should be able to detect verbatim plagiarism between a pair of documents. This detection should only indicate if the text pair should be manually read in order to confirm the plagiarism.

1.5 Thesis Hypothesis

By using word n-grams a efficient technique for automated verbatim plagiarism detection can be developed. This technique should work on digital documents written in English.

1.6 Thesis Contribution

The main contribution of this thesis will be a technique and a procedure for automated textual plagiarism detection. This procedure will be used in the development of the DOCODE system. Also, the prototype developed from the proposed technique will participate in the international Workshop and Competition on automated plagiarism detection, PAN 2010 [30].

1.7 Thesis Structure

On the following chapter, the bibliographic review is presented where the state of the art in automated plagiarism detection and search space reduction techniques are reviewed.

On chapter 3, the main contribution of this thesis is presented, where the problem definition and the plagiarism detection strategy are described.

Then, on chapter 4, an experiment design is presented along with benchmark algorithms for comparison purposes. Next, the experiment is conducted in order to determine accurate settings for the technique and for further evaluation.

Afterwards, on chapter 5, the results for the algorithm and proposed technique are discussed.

Finally, on chapter 6 the main conclusions are presented, where the main findings and contributions are highlighted, as well as the future work and lines for research are discussed.

Chapter 2

Conceptual Framework

In this chapter, a conceptual framework and different topics about plagiarism detection are introduced.
In this context, it is necessary to differentiate the task of plagiarism detection from author recognition. The latter is mainly done by investigating facts around the work itself in order to determine the actual Author, and by identifying style markers that could lead to identify differences in writing style. The former is about whether or not a work has been copied, following one of the plagiarism forms, described in section 1. In plagiarism detection two main topics can be described: on the one hand, intrinsic plagiarism detection, task aimed at discovering plagiarism examining only the work or document in question. On the other hand, external plagiarism detection, where the suspicious documents are compared against a set of possible references.

2.1 Plagiarism

First, definitions about plagiarism are introduced. Then, implications of plagiarism in academia and it's current status are reviewed. Finally, some reasons about why plagiarism could be considered as a positive practice within academia [16] are discussed.

2.1.1 What is Plagiarism?

According to the Collins Dictionary of the English Language [15], plagiarism is *the act of plagiarising*, which means *"to appropriate (ideas, passages, etc) from (another work or author)"*. Plagiarism involves literary theft, stealing (by copying) the words or ideas of someone else and passing them off as one's own without crediting the source.

Also, in [28] other definitions are given, such as:

> The term plagiarism is usually used to refer to the theft of words or ideas, beyond what would normally be regarded as general knowledge. This is the spirit of the definition of plagiarism adopted by the Association of American Historians, who describe it as "the misuse of the writings of another author...including the limited borrowing, without attribution, of another's distinctive and significant research findings, hypotheses, theories [...] or interpretations" [13].

It is clear that plagiarism refers to a complex topic, that usually involves first analyzing the context and choosing a proper definition.

2.1.2 Plagiarism in academia

Plagiarism in academia is rising and multiple authors have worked to describe this phenomena [16, 17, 28]. As commented by Hunt [16], "Internet Plagiarism" is referred sometimes as a cataclysmic consequence of the "information technology revolution", as it proves to be a big problem in academia. In [28], plagiarism is analyzed from various perspectives and considered as a problem that is growing bigger over time. In [17] the author analyzes different statistical data and the implications of the "IT age". He then discusses the significant number of students engaged in inappropriate academic practices. As listed in [16], main plagiarism forms, in the case of students, can be described as:

1. Stealing material from another source and passing it off as their own, e.g.
   (a) buying a paper from a research service, essay bank or term paper mill (either pre-written or specially written),
   (b) copying a whole paper from a source text without proper acknowledgement,
   (c) submitting another student's work, with or without that student's knowledge (e.g. by copying a computer disk).
2. Submitting a paper written by someone else (e.g. a peer or relative) and passing it off as their own.
3. Copying sections of material from one or more source texts, supplying proper documentation (including the full reference) but leaving out quotation marks, thus giving the impression that the material has been paraphrased rather than directly quoted.
4. Paraphrasing material from one or more source texts without supplying appropriate documentation and references.

To tackle this problem, one approach is to try to detect plagiarism. Different methods involving computer aided plagiarism detection have been under research [9, 10, 14, 19, 23, 36, 37],

from which different system for automatic plagiarism detection have been developed.
However, different ways for neutralizing such detection systems have been presented. Such methods usually involve modifying the text in such way that the presentation of the document remains the same, but the underlying code is different and normally this differentiation render the detection systems useless. For example, as presented in [27], one can substitute a Latin *'o'* for a Russian *'o'*. In terms of presentation it reads the same. But the `ASCII` code, the code that the detection program reads, is different.

This form of ethic fault is being exploited for example using a software called "Anti Plagiat Killer", as mentioned in [27]. The steps for accomplishing this, would be; first a student gets the instructions. Then, he writes the document. Next, by using one of these softwares, for example "Anti Plagiat Killer", he gets the document processed with the real text "hidden". Finally he submits the resulting work, confusing the automated plagiarism detection system thus getting undetected. For these cases, the automatic plagiarism detection system should try to detect these modifications in order to correctly operate.

2.1.3 Discussion: why plagiarism could be a good thing?

As discussed in [16], plagiarism could be good on certain things. The arguments and ideas behind this perspective are as follow:

The institutional rhetorical writing environment (the "research paper," the "literary essay," the "term paper") is challenged by plagiarism, and that's a good thing. This is because the idea of measuring a students learning by evaluating it's capacity on writing is questionable, so it's necessary to rethink which environment variables teachers should consider at this evaluation.
Also, the institutional structures around grades and certification are challenged by plagiarism too, and that's a good thing. The pressure around students to get better grades often diminish the importance of the task being done, motivating students focus on the goal rather than working hard and honestly.
And this is also related to the model of knowledge held by almost all students, and by many faculty – the tacit assumption that knowledge is stored information and that skills are isolated, asocial faculties. Plagiarism put in question this idea as it correct to assume the learning process and the writing of students are conditioned by situations or expectations, and it is hard to accept those could be reproduce in a educational institution.

The author [16] emphasizes the reason plagiarism could be good is that it opens questions that could lead to the better understanding of today's teachings methods and how the knowledge is constructed. So certainly this discussion is not simple and often conducts to opposing ideas, but this different perspective brings a new way to understand plagiarism and it's implications.

2.2 Automated Plagiarism Detection

Nowadays with a large set of possible sources for plagiarism such as the Web, it is important to use the technology available to aid in it's detection. So, different techniques are used for automated plagiarism detection. First, the documents needs to be characterized in order to be able to get it's information and to determine if a plagiarism case exist. A set of characteristics can be used from a text in order to proceed with a detection method [12]:

1. Uses of vocabulary. The used vocabulary is compared against other texts written by the same author. The difference of the two sets of words used can be helpful to determine if the author incurs in some kind of plagiarism.
2. Changes of vocabulary. The changes in the vocabulary used within a single text is analyzed, as it is expected that different students use words in different ways.
3. Incoherent text. The document is analyzed in order to determine whether it contains incoherent passages or not.
4. Punctuation. It is unlikely two different authors use the exact same punctuation style, such as the use of periods.
5. Common spelling mistakes. It is unlikely different authors will make the same mistakes.
6. Distribution of words. It is unlikely the same words distribution, the frequency words are used within a single text, are equal amongst independent texts.
7. Syntactic structure of the text. If two texts share exactly the same syntactic structure, it may be due to plagiarism.
8. Long sequences of common text. Word sequences comparison can be used to determine plagiarism.
9. Order of similarity between texts. If a large set of common words or sentences are found to be common, a high probability of plagiarism can be assumed.
10. Frequency of words. The frequency of words can be analyzed in order to determine plagiarism possibility.
11. Preference for the use of short or long sentences. The preference of the author to use whether short or long sentences can be helpful to analyze plagiarism cases.
12. Readability of written text. Different indicators of legibility, such as the number of complex words, can be used to help determine plagiarism.
13. Dangling references. If there are errors in the citation or in the reference section, it is necessary to check further for a possible plagiarism case.

These characteristics must be first understood before any method can be constructed around them. For example, it is expected a student to widen it's vocabulary over time, with the consequent change of the uses of vocabulary. This consideration could be taken into account for a plagiarism detection method development. The same can be said about punctuation, spelling mistakes and other characteristics.

Furthermore, different methods for plagiarism detection can be categorized as follow [25]:

1. Exhaustive comparison between a suspicious document and a set of references.
2. To characterize a fragment of a document to be search over the Web.
3. To compare characteristic within a single text, using style properties.

The first category is known as external plagiarism detection [30], which will be considered as the main focus in this research. In second category, a given suspicious document is chopped into a set of queries for use with web search engines. The result set obtained is used as sources candidates of the given suspicious document whose outcome can be considered as input for first category algorithms. In [8] a tool for this task is proposed. Finally, the third category known as intrinsic plagiarism detection [30] is described in the following subsection.

2.2.1 Intrinsic Plagiarism Detection

When comparing texts against a reference set of possible sources, comes the complication of choosing the right set. And now more than ever, with the possibilities the Internet bring to plagiarists, this task becomes more complicated to achieve. For this, it is possible to use intrinsic plagiarism detection. This approach only analyzes the suspicious document, thus takes not into consideration a set of references.

The writer style can be analyzed within the document and an examination for incongruities can be done. As in [40], the complexity and style of each text is analyzed based on certain parameters. These parameters are:

1. Text statistics, such as the number of commas, question marks, word lengths, or any other information which operate at the character level.
2. Syntactic features, such as sentence lengths and use of function words, which measure writing style at the sentence level.
3. Part-of-speech features to quantify the use of word classes, such as the number of adjectives or pronouns.

4. Closed-class word sets to count special words, such as the number of stopwords, foreign words, "difficult" words.

5. Structural features, such as paragraph lengths or chapter lengths, which reflect text organization.

Based on these features, intrinsic plagiarism detection methods are build in order to characterize the writer's style. Then, differences within the write's style are analyzed in order to determine if the style has changed enough to identify a possible case of plagiarism. This could be, for example, the case when a student copy a paragraph from the Web. It is possible that the student's writing style does not match the style of the copied paragraph. By identifying this difference it is possible to believe that the student had plagiarized.

Meyer zu Eissen et al [40] introduced a new feature for detecting style changes: Averaged Word Frequency Class. This feature is calculated as shown in equation 2.1:

$$c(w) = \lfloor \log_2(f(w^*)/f(w)) \rfloor \tag{2.1}$$

where the Averaged Word Frequency Class is defined as *c(w)*, *f(w)* being the frequency of a word w, and w^* denotes the most frequently used word.

According to the authors, a document's averaged word frequency class is useful as a feature to identify style complexity and the size of an author's vocabulary, thus proving promising for the construction for an automated intrinsic plagiarism detection method.

Stamatatos [38] presented a new method for intrinsic plagiarism detection. As described by it's author, this approach attempts to quantify the style variation within a document using character n-gram profiles and a style change function based on an appropriate dissimilarity measure originally proposed for author identification. Style profiles are first constructed, using a sliding window. For the construction of those profiles the author proposed the use of character n-grams. These n-grams are used for getting information on the writer's style. The method then analyzes changes on the profiles to determine if a change is significative enough to indicate another's author style. Stamatatos' approach prove to be the best amongst the four approaches presented in PAN Workshop and Competition'09 [30] at detecting intrinsic plagiarism.

Seaward and Matwin [34] introduce Kolmogorov Complexity measures as a way of extracting structural information from texts for Intrinsic Plagiarism Detection. They experiment with complexity features based on the Lempel-Ziv compression algorithm for detecting style shifts within a single document, thus revealing possible plagiarized passages. They also participated on PAN Workshop and Competition'09 [30], and their results can be compared against other approaches on

the competition web site [1].

Intrinsic plagiarism detection is becoming more important as to define the possible sources, to utilize external plagiarism detection methods, is becoming more difficult. Now, part of the PAN Competition and Workshop [30] is dedicated to promote the research in this particular area.

It is important to note that by the use of an intrinsic method for plagiarism discovery, it is not demonstrated that a paragraph or a part of the document is being copied, because there is no reference to compare to. Therefore, this kind of plagiarism detection category is only indicative and should be used in conjunction with human supervision. Nevertheless, intrinsic plagiarism is useful when trying to discover originality or authorship of a document.

2.2.2 External Plagiarism Detection

When comparing a suspicious document against a collection of possible sources, it is tried to identify the sentences, paragraph or ideas that have been copied. This is called external plagiarism detection [30], and multiple efforts are being oriented in this area.

Before the comparison between each possible source and the suspicious document can be computed, an important obstacle must be resolved. This task consists in defining and gathering the possible sources, and this is becoming more and more complex as the technology becomes more available. In [8], the suspicious document is chopped into queries and web search engines are used to obtain a set of candidates sources. This approach helps tackle this problem but, as the authors in [8] conclude, more work is needed.

Another issue to be considered, is when the collection of possible sources become too large. In fact, these could be thousands of documents. In [30], PAN Competition and Workshop 2009, the external part of the competition, and now merged with the training corpus, consider a set of 14.428 possible sources. To tackle this problem, it is possible to reduce the search space using different data mining techniques. A generic process for automatic detection techniques is illustrated in Figure 2.1. Next, different methods for comparing documents are reviewed.

Seo and Croft [35] introduced in their work an approach for local text reuse detection. They propose *DCT fingerprinting*; a sequence of hash values of words can be considered as and transformed into a discrete time domain signal sequence. With this, they showed that fingerprinting using this approach a more robust data reduction of the document, for it's later comparison, can be achieved.

In [9] an automated system for copy detection was introduced, named COPS after Copy Protection System. The system detects document overlap based on sentence and string matching but it cannot find partial sentence copy. Then, in [36] another system was introduced. SCAM, for

[1] `http://www.uni-weimar.de/medien/webis/research/workshopseries/pan-09/competition.html#results`

Stanford Copy Analysis Method, uses a Relative Frequency Model (RFM) to find subset copies. RFM is the first asymmetric model, which implies that the model takes into consideration the suspicious document as suspicious, and the source as the reference.

In [37] a mechanism to detect overlap fragments based on information retrieval techniques is introduced. This system is called CHECK, and builds an index called structural characteristic to perform the detection. The system takes into consideration that documents from different topics should not be compared, thus probing more scalable on the number of documents to compare.

Another approach is presented in [24], where the use of word n-grams for plagiarism detection is explored. The use of n-grams give some flexibility to the detection, as reworded fragments could still be detected. In particular, in [2] the tri-gram structure is found to be the most effective in this task. This method is possible because the common word n-grams between two documents are usually a low percentage of the total number of n-grams of both text, as shown in Table 2.1. Due to this, n-grams could probe promissory for plagiarism detection techniques.

In [23], Lyon et al extended their work and the Ferret system was implemented, which uses this approximation to detect plagiarism. A distance is calculated between the documents, based on the word n-grams found in common. The results indicate that this structure is useful and provides flexibility at detecting plagiarism with modifications of words.

Table 2.1: Statistics from a TV news corpus, the Federalist papers and the Wall Street Journal corpora, showing the predominance of unique trigrams. From [24].

Source	Corpus size in words	Distinct trigrams	Singleton trigrams	% of trigrams that are singletons
TV News	985,316	718,953	614,172	85%
Federalist Papers	183,372	135,83	118,842	87%
WSJ	972,868	648,482	556,185	86%
	4,513,716	2,420,168	1,990,507	82%
	38, 532,517	14,096,109	10,907,373	77%

Table 2.1 shows statistical data from different documents remarking the high percentage of unique tri-grams found between them. Therefore, a distance based on these word tri-grams could be computed to construct an indicator for plagiarism.

In [19] another system is introduced, PPChecker. The system analyzes the sentences of a suspicious document to determine plagiarism. To tackle the problem of rewording, PPChecker uses Wordnet [2] for synonym recognition. With this approach the system not only use the vocabulary of each text: it also consider the synonyms found to detect one of the following copy cases:

1. Exact copy.

[2] `http://wordnet.princeton.edu/`

2. Copy with word insertion.

3. Copy with word extraction.

4. The use of synonyms.

Bao et al. in [18] and then in [1], proposed to use a Semantic Sequence Kernel (SSK)([33]), and then using it into a traditional Support Vector Machines (SVMs) formulation based on the Structural Risk Minimization (SRM) [7, 39] principle from statistical learning theory, where the general objective is finding out the optimal classification hyperplane for the binary classification problem (plagiarized, not plagiarized). Likewise, other approaches solve the same classification problem by using Self Organizing Feature Maps (SOFM) [21], with promising results in the classification performance.

In [10] a method using Singular Value Decomposition (SVD) [6] is proposed. This approach, as noted by the author, uses the denominated Latent Semantic Analysis (LSA) [6], a technique to infer the latent semantic associations and subsequently determine the document similarity. In the process, LSA uses SVD, factorizing the matrixes containing the weighted occurrences of the phrases of each document. The author found this approach overcomes other methods at detecting plagiarism cases.

In [14] a new general method for automatic external plagiarism detection is proposed. It consists of two main phases: the first one aim to reduce the search space. For a suspicious document, only the source documents selected in this phase are marked for further investigation. On the second one, each pair of documents are exhaustively investigated for plagiarized passages, by comparing 16 bit strings. This way a matrix is constructed, indicating each match between those strings. Then a contiguity distance is computed in order to determine the size of the plagiarism cases. "Encoplot", as they named the method, had them won the external plagiarism detection task of the PAN Workshop and Competition of 2009.

Kasprzak et al. in [20] used word n-grams, with the value of n ranging from 4 to 6, to find similarities within documents, combined with inverted indexes to accelerate the computation. Their approach resulted in a fast algorithm capable of determining which pair of documents should be classified as plagiarized and also capable of finding the passages. Their work showed promising results in PAN Workshop and Competition of 2009 at the task of discovering monolingual plagiarism with different obfuscation levels.

Basile et al. [5] also participated in the named competition. Their proposed approach is divided in three steps: the first aimed at reducing the search space, the second aimed at finding matches of features between text, and finally the step where the matches found are interpreted and rectified in order to indicate the plagiarized passages. For the first step, they recoded the word segments into a sequence of numbers; each word was represented with it's character length. Then a distance is calculated based on 8-grams of these representations. The first ten source documents, ranked by this distance, are selected. For the second step, a T9 like representation is used for coding

the words. By looking for matches of sequences of these word representations, the third step tries to interpret the results and indicate the plagiarized passages found. This approach took third place on the PAN Workshop and Competition of 2009 [30].

2.2.3 Cross-Lingual Plagiarism Detection

Another topic of discussion is plagiarism when translation from different language is involved. Cross-lingual plagiarism detection considers suspicious documents and source documents, similar to external plagiarism detection, but in this case the fingerprints need to be worked around the language difference.

Pouliquen et al [31] analyzes an approach considering the EUROVOC[3] thesaurus. EUROVOC is a multilingual and multidisciplinary thesaurus the European Union maintains with terminology mainly focused on parliamentary affairs. The authors used the thesaurus to map each document in a language independent form, and then calculate cosine similarity measure between the texts vectors. Experimenting with English, French and Spanish, their approach showed encouraging results [31].

In [3] discussion and analysis on cross-lingual plagiarism detection is presented. They based the analysis on a statistical bilingual dictionary, created on the basis of a parallel corpus which contains original fragments written in one language and plagiarized versions of these fragments written in another language.

Another approach is presented by Ceska et al. in [11]. Their method, called "MLPlag" for multilingual plagiarism detection, is based on word position analysis. They first utilize the EuroWordNet thesaurus for transforming the words into language independent forms, and then proceed with examination in search for plagiarism.

Potthast et al. [29] introduce and analyze different methods for uncovering this kind of plagiarism. They reviewed heuristic multilingual retrieval of potential source candidates for plagiarism from the Web, and methods for comparing documents across languages. The methods for comparing the documents are based on cross-language character n-gram (CL-CNG), on cross language explicit semantic analysis (CL-ESA) and on cross-language alignment based similarity analysis (CL-ASA). They experiment on a large-scale comparative evaluation, reporting their findings.

2.3 Reducing Search Space

One of the issues to be resolved in external plagiarism analysis and detection is the number of source document candidates. When the task is to detect plagiarism between a small set of suspicious against

[3] http://eurovoc.europa.eu/

a small set of source documents, it could be simply achieved by comparing every suspicious against every source. The problem appears when the universe of possible sources is not well defined, or the set of documents is too large, so performing an exhaustive comparison amongst all documents could require a substantial amount of time. In this case the approach needs to be modified, and those changes usually consists in adding a step in the process of plagiarism discovery: the search space reduction.

The aim of this step is to effectively and efficiently identify which texts are possible sources of plagiarism, if any. Usually multiple statistical tools are used in order to reduce the computational time required for computing a large corpus of documents while trying to maintain accuracy at determining which sources need to be discarded. Figure 2.1 shows the complete process.

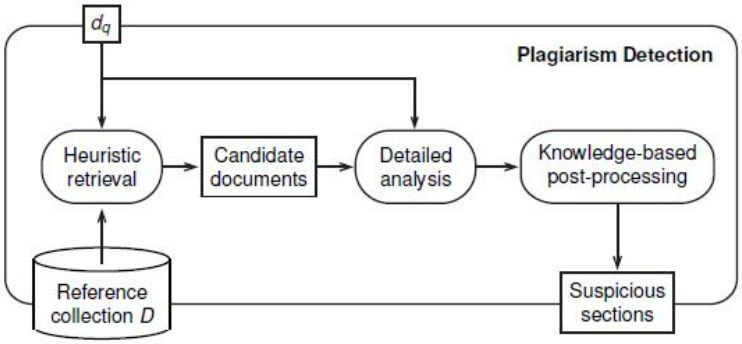

Figure 2.1: Generic retrieval process for external plagiarism detection [30].

In the process Figure 2.1 shows, a Reference collection D is considered. When a suspicious document needs to be analyzed, an heuristical retrieval is conducted to determine which texts from D are promising candidates. Then, a detailed analysis is conducted between the suspicious document and the promising sources. Finally the results are post-processed and archived.

Different concepts regarding this increasingly important task are described next.

2.3.1 Vector Space Model and Cosine Similarity Measure

Before any method can be used to establish some kind of similarity or distance measure between documents, the texts need to be characterized in some way. In [32] a method is introduced. It consists in characterizing a text in a vector. It's values are constructed based on term frequency and then this value is weighted by a factor denominated inverted document frequency. TF-IDF, as is known, takes into account the number of occurrences of the words in the text (TF), and then it uses information from the whole set of documents to be compared, specifically the number of

documents that contain said term (IDF).

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2.2}$$

Term frequency refers to the number of times the t_i term appears in the document d_j. In Equation 2.2, $n_{i,j}$ refers to this count, whereas $\sum_k n_{k,j}$ refers to the total number of terms in the document.

$$idf_i = log\frac{|D|}{|d : t_i \in d|} \tag{2.3}$$

The *Inverse Document Frequency* refers to the importance of a certain term considering it's appearances in the whole set of documents. So, a term that appears in all documents, probably has little information to difference the text, whereas a term only appearing in a few documents can be of interest. In Equation 2.3 $|D|$ refers to the total number of documents. $|d : t_i \in d|$ indicates the number of documents where the term t_i appears at least once.

$$(tf - idf)_{i,j} = tf_{i,j} * idf_i \tag{2.4}$$

Finally, in Equation 2.4 the *term frequency - inverse document frequency* is calculated. With these values a matrix can be constructed, representing each term importance for each document.

Once the text has been characterized and the matrix containing the values have been computed, the distance or similarity measure based on the cosine between the vectors can be calculated. The formula is shown next:

$$\cos \measuredangle(\mathbf{x}_A, \mathbf{x}_B) = \frac{\mathbf{x}_A \cdot \mathbf{x}_B}{\|\mathbf{x}_A\| \, \|\mathbf{x}_B\|} \tag{2.5}$$

In Equation 2.5, two vectors, $\mathbf{x}_A$ and $\mathbf{x}_B$, are used to calculate their similarity. In this case, each vector represents a document, with it's values being, for example, the ones given by 2.4. So, the document vector is constructed using TF-IDF, and the cosine similarity measure is used to compare two documents. Given that the values from In Equation 2.4 cannot be negatives, the resulting value for the cosine will range between 0 and 1; values close to 1 indicating similarity and the ones close to 0 indicating difference. It is important to note, that while the resulting values from TF-IDF are not normalized, the cosine distance does not take into account the magnitude of the vectors, thus this information is lost.

2.3.2 Kullback-Leibler distance

In [4] a different distance is calculated. This distance is named Kullback-Leibler divergence [22], and is calculated as shown in Eq. 2.6:

$$KL_d(P||Q) = \sum_{x \in X} (x) \log \frac{P(x)}{Q(x)} \tag{2.6}$$

In Eq. 2.6, **x** represents a feature vector, whereas P and Q are the probability distributions of the two documents. These distributions could be, for example, TF-IDF vectors for such documents, or TF vectors. If TF-IDF is selected, the Equation 2.4 is first computed. Each probability distribution, namely *P, D* in this case, is constructed based on the top terms from the resulting vector of TF-IDF. The idea is that the list of selected terms represents each document information.

In [4] an investigation is conducted to determine how to represent those probability distributions. They found the best results are obtained using TF-IDF. This investigation where conducted to find the best choice for representing a document, in the context of automated plagiarism detection.

2.4 Free and Commercial Tools and Systems for Plagiarism Detection

Currently, there are several tools and systems for automated plagiarism detection. A few of them, as Turnitin, are offered as a service, whereas others, like SIM, can be downloaded and run on a computer. A short description of a few of them is given.

- Turnitin
  $http://www.turnitin.com/$ Turnitin is a commercial company that offers services for plagiarism detection. It check student's work against continuously updated databases. It currently has more than hundred million students papers, over twelve millions crawled web pages and access to magazines and newspapers.
- CopyTracker
  $http://copytracker.ec-lille.fr/$ CopyTracker is a free software for home use that checks document for plagiarism passages. It compares the suspicious document against possible source documents provided by the user and also search over the Web looking for potential sources.
- EVE2
  $http://www.canexus.com/$ EVE2 is a commercial tool that search over the Web for possible

sources of a suspicious student paper. It returns the URLs it finds, and also a full report to the teacher is also provided.

- PlagiarismDetect.com
  http : *//www.plagiarismdetect.com/* PlagiarismDetect.com is a commercial tool that search over the Web for possible sources. Work similar to EVE2.

- Glatt Plagiarism Services
  http : *//www.plagiarism.com/* Glatt Plagiarism Services offers three softwares; the first is a Tutorial Program for help educating students about what is plagiarism and how to avoid it, the second one is a Screening Program to detect plagiarism in documents, whereas the third is also a Screening Program for detection of inadvertent instances of plagiarism.

From a functional perspective, Turnitin has three important characteristics. The first is that when comparing suspicious documents, it uses all the previously submitted papers for comparison purposes. This is important because it has a growing database of sources, that grows as more detections are conducted. The second characteristic is that it considers possible sources from the Web. It crawls webpages to feed the database, therefore when comparing documents it can detect plagiarism from the Web. Finally, Turnitin is a established company with trusted algorithms for plagiarism detection. This makes it as the company of choice for getting plagiarism detection services, as the algorithms work in an effective manner.

Considering these functional characteristics, Turnitin is the model to follow. The proposed method for plagiarism detection has taken into consideration these important functions. It is designed to handle large amounts of documents, it can computed texts retrieved from the Web, and it is tested to be reliable and effective.

2.5 Chapter Summary

In this chapter a general review and state of the art algorithms about plagiarism and plagiarism detection has been presented. Plagiarism is an important problem worldwide, and it could be tackled with the help of technology. In the field of automated plagiarism detection, specifically in text plagiarism, different methods have been developed. Promising results have been achieved with the use of word n-grams for finding plagiarized passages. Also, different services from companys, such as Turnitin, have been briefly presented, and important functions of these services highlighted. The proposal of this thesis takes into account the use of word n-grams and the characteristics described for these services.

Chapter 3

Plagiarism Detection Strategy Proposal

In this chapter the designed detection strategy is proposed. This strategy is based in the comparison of word n-grams [2, 24] and is aimed to detect verbatim plagiarism. The method removes "stopwords" and for speed purposes, it considers approximation on selecting samples within the documents. The algorithm output is an indicator, whose interpretation classifies the pair of document as to have plagiarism and should be further investigated, or as the pair does not shows signs of plagiarism.

3.1 Technical Background

First, concepts used for the development of the proposed method are going to be described. In the Technical Background the three main ideas are introduced, namely, the *Word N-Grams*, *StopWords* and the used notation.

3.1.1 N-Grams

The proposed detection strategy uses word tri-grams and word two-grams as is basic element for comparison. A word n-gram is a structure where consecutive words are grouped in a set of n, maintaining the order. For example:

```
Gravitation is not responsible for people falling in love.
```

would be chopped in seven different tri-grams:

1. Gravitation is not
2. is not responsible
3. not responsible for
4. responsible for people
5. for people falling
6. people falling in
7. falling in love.

These basic structure where studied and used in [24] and in [2] with results indicating their effectiveness when building an algorithm based on them for plagiarism discovery.

3.1.2 StopWords

Second, the proposed method removes "stopwords" from the text as a preprocessing step. "Stopwords" are words that are common amongst text and one can assume little information can be retrieve by considering them. Examples of "stopwords" are prepositions and articles. In addition, while plagiarizing one can think that it is easier for the guilty to change or modify these words rather than modifying the words important for the topic.

The example above, before constructing the set of n-grams, could be processed as follow:

Before the removal:

```
Gravitation is not responsible for people falling in love.
```

After the removal:

```
Gravitation responsible people falling love.
```

3.1.3 Notation

In the following, let $\mathcal{V}$ be a vector of words that defines the vocabulary to be used. A word will be represented as w, as a basic unit of discrete data, indexed by $\{1, ..., |\mathcal{V}|\}$. A document d is a

sequence of S words ($|d| = S$) defined by $\mathbf{w} = (w^1, ..., w^S)$, where w^s represents the s^{th} word in the message. Finally, a corpus is defined by a collection of $\mathcal{D}$ documents denoted by $\mathcal{C} = (\mathbf{w}_1, ..., \mathbf{w}_{|\mathcal{D}|})$.

The use of these ideas needs to be materialized in a concrete method. Chopping the text into segments of a certain fixed length, l, and then calculating the number of common n-grams for each pair of these segments between texts, it is how the proposed method works. If the number of common n-grams in one pair of these segments, one from the first document and the second from the other, is greater than a parameter k, then a similarity indicator is increased. When finishing comparing every segment pair, the named indicator is given.

This method is aimed at trying to uncover verbatim plagiarism by indicating if the document pair should be investigated further. In the next subsection the strategy is detailed.

3.2 Automated Plagiarism Detection Proposal

In this section the proposed method is introduced. For automated plagiarism detection, the method needs to consider the complete process; from documents loading into memory, document information retrieval, including preprocessing and fingerprinting, computing the comparison and reporting the obtained results.

3.2.1 Detection strategy and interpretation

The general procedure of the strategy is shown in Figure 3.1.

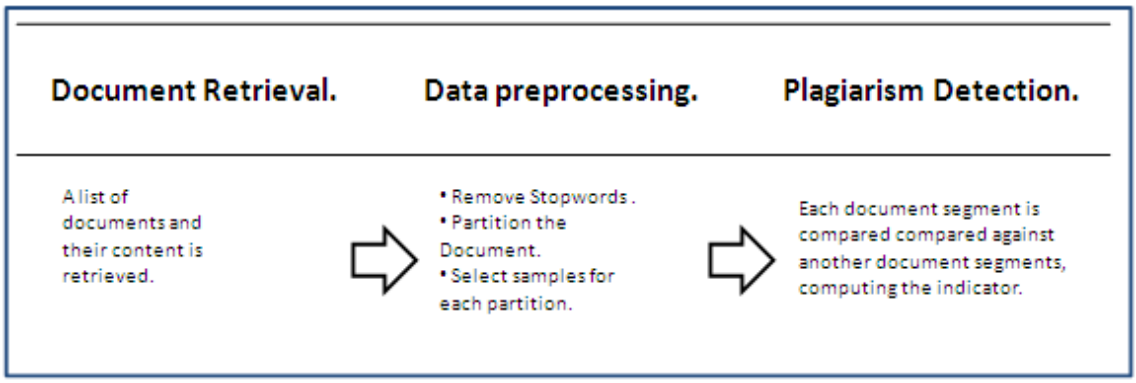

Figure 3.1: General procedure for verbatim plagiarism uncovering.

- Data preprocessing

In this step, the necessary preprocessing is done. This consist mainly in taking the words of the document and considering them as "tokens". Second, all characters are transformed to lowercase, and stopwords are remove in order to consider only words that offer the most information and to reduce the number of n-grams, and thus reducing the number of comparisons. Figure 3.2 shows the steps described above.

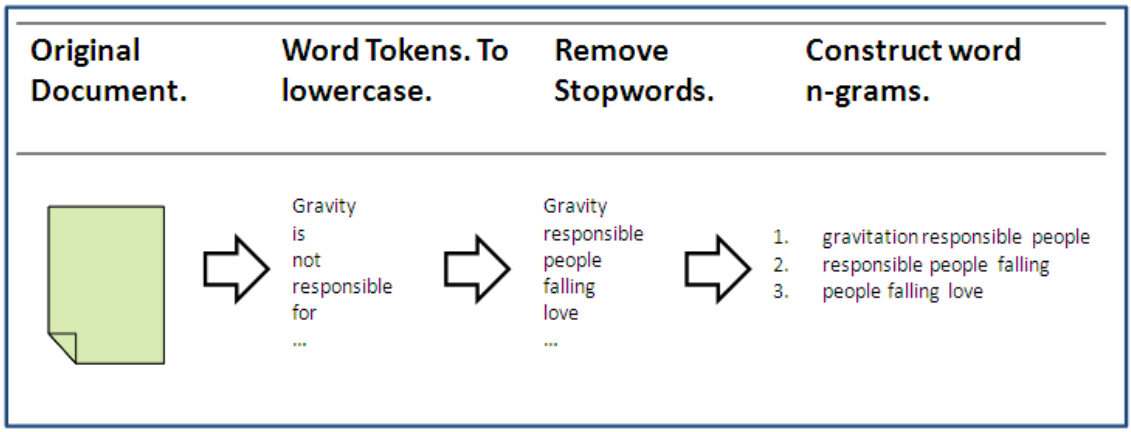

Figure 3.2: Firsts steps: preprocessing of documents starts, taking a document and building it's set of word n-grams.

- Plagiarism detection

In this step, once the data to be processed is ready, the algorithm for determining a similarity value begins. Algorithm 3.2.2 first evaluates an $\text{sMatch}(\mathbf{t}_i, \mathbf{t}_j, s \geq 1)$ algorithm which returns true whether at least one n-gram from $\mathbf{t}_i$ matches one n-gram from $\mathbf{t}_j$. Also a variation of previous matching method is used within the segments of n-grams. Condition $\text{sMatch}(\kappa_i, \kappa_j, s \geq \theta_1)$ states that at least θ_1 n-grams must match in between all segments κ_i and κ_j. If this is hold, the next condition $\text{sMatch}(\mathbf{t}_i, \mathbf{t}_j, s \geq \theta_2)$ is associated to find whether at least θ_2 n-grams matches between $\mathbf{t}_i$ and $\mathbf{t}_j$. In general terms, this procedure helps on reducing the search space, and improving the algorithm in both execution time and hardware requirements. By using these constraints, it is possible now to go into a further algorithm for finding the needed offset and its length. Figure 3.3 shows the steps for computing the indicator.

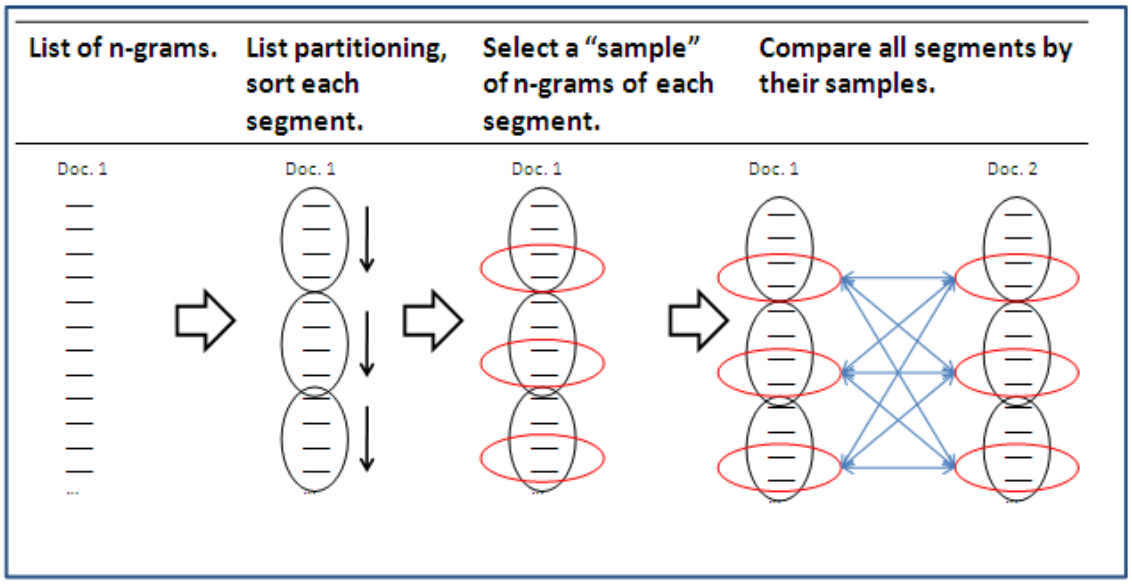

Figure 3.3: Performing the comparison: once the preprocessing is done, the comparison is computed.

3.2.2 Pseudo code

Figure 3.1, Figure 3.2 and Figure 3.3 show how the detection is being made; the underlying code of those steps is presented next.

Algorithm 3.2.1: PREPROCESSDOCUMENT

Data: $\mathbf{d}_i, n, k, m$

1 REMOVESTOPWORDS($\mathbf{d}_i$);
2 $\mathbf{t}_i \leftarrow$ GENERATENGRAMS($\mathbf{d}_i, n$) ;
3 $\mathbf{k}_i \leftarrow$ GENERATEKNGRAMS($\mathbf{t}_i, k$);
4 $\mathbf{k}_i^* \leftarrow$ SORT($\mathbf{k}_i$, SORTSTRATEGY) ;
5 $\kappa_i \leftarrow$ SELECTMLASTNGRAMS($\mathbf{k}_i^*, m$) ;
6 return $(\kappa_i, \mathbf{t}_i)$;

As presented in algorithm 3.2.1, new methods are introduced for the processing, such as the GENERATENGRAMS function that takes a given document d_i and returns a set of n-grams with the structure $(w_i, w_{i+1} \ldots, w_{i+n}), \forall i \geq 1, n \leq S$. Function GENERATEKNGRAMS, generates groups of length k using all n-grams. Then, a SORT algorithm is used within segments, with a specific sorting strategy. In this research, an Alphabet sorting strategy and a Term Frequency sorting strategy where used as a variation on the proposed algorithm. Finally, a SELECTMLASTNGRAMS function,

as specified in its name definition, selects only the last m n-grams within the segment.

Algorithm 3.2.2: ApproximateComparison

```
Data: κ_i, κ_j, t_i, t_j, θ_1, θ_2
if sMatch(t_i, t_j, s ≥ 1) then
    if sMatch(κ_i, κ_j, s ≥ θ_1) then
        if sMatch(k_i, k_j, s ≥ θ_2) then
            return true ;
        end
    end
end
else
    return false;
end
```

Once documents d_i and d_j are processed in n-grams and segments of n-grams, t_i, t_j and κ_i, κ_j respectively, a set of conditions are evaluated in order to set the relation that document d_i has with document d_j, that is, if they are somehow related (algorithm 3.2.2 returns true), or if it is not worthy to keep finding further relationships (algorithm 3.2.2 returns false). In this sense, this is an approximated finding procedure that considers both n-grams and their k segments to decide if there is enough information to classify as plagiarism or not.

3.3 Development

In this section a brief description of the developed classes and their methods will be given. Three main classes were used, the first for loading the documents to be compared and to control the entire process, the second an auxiliary class with useful methods for preprocessing, and the third receiving the word n-grams for both documents and computing the plagiarism indicator.

3.3.1 Classes and Methods

- *Dist*
  This class takes as inputs a list of suspicious documents and a list of possible sources. It loads into RAM memory a vector representation for each document, containing all words maintaining their original order. Then it invokes the main method of the *Approximate Comparison* class, taking it's output - the resulting indicators - and saving it to a XML file.

  – Inputs
  List of suspicious documents.
  List of source documents.

  - Outputs
    XML file. This file contains the results of the comparison for each suspicious document against each possible source.

- *Services*
  This class provides useful methods for the algorithm. The main methods are for constructing the word n-grams of a given vector, for partitioning the vector, to remove stopwords, for sorting the vector based on different sorting strategies and to obtain a sample of the partition.

  - Methods

    * *RemoveStopwords*
      This method receives an indexed list of words, removes the stopwords and then return the new list.
    * *GenerateNGrams*
      This method receives an indexed list of words and a parameter, N, and return the list of corresponding $N-Grams$.
    * *GenerateKNGrams*
      This method receives an indexed list of $N-Grams$ and a parameter, K, and does a partitioning based on it. This partitioning returns a set of list, each representing a partition.
    * *Sort*
      This method receives a list of indexed words and a parameter indicating the strategy to be used. The methods sorts the list based on the selected strategy and then return the new list.
    * *SelectMLastNGrams*
      This methods receives a list of $N-Grams$ and a parameter, M, and return the last m $N-Grams$ of it. In conjunction with *Sort* this method takes "samples" of a vector.

- *Approximate Comparison*
  This class computes a pair of documents. First, it preprocess the data, removing stopwords, constructing the word n-grams and partitioning them. Then, it uses a strategy for taking "samples" of each segment and then computes the comparison, thus obtaining an indicator that is finally returned.

  - Inputs
    Two vectors representing the documents to be compared.

  - Outputs
    A number representing the plagiarism indicator.

Chapter 4

Experiments

In this Chapter a description of a conducted experiment is given. First, the general idea for the experiment is presented. Then the data used is characterized; the documents are extracted from the PAN Workshop and Competition 2009 corpora, [30]. This corpus contains suspicious documents and possible references as well. The plagiarism cases included are mainly computed generated. A more detailed explanation is given in Section 4.1. Next, the strategy implementation is described and finally the experiment is conducted with others approaches for comparison purposes. There are three algorithms implementation, and with each one of them, three variants are tested. In total, the nine runs are compared, including as reference three algorithms based on levenshtein distance.

4.1 Experiment Corpus Characterization

PAN Workshop and Competition [30] corpora is chosen as the resource depot for suspicious and source documents. The training corpora for 2010 PAN Competition consist of 14.428 suspicious documents and 14.428 source documents. These numbers are for the external detection task, that consist in determining if a suspicious document presents a plagiarism case from one or more reference documents. Also, the annotation for each plagiarism case is given, therefore the participants can tune and test their approaches. Statistics from the corpus are as follow:

1. 14.428 Source documents.
2. 14.428 Suspicious documents.
3. Plagiarism cases with different obfuscation levels: 33% none obfuscation, 33% with low obfuscation and 33% with high levels of obfuscation.
4. 90% of the cases are monolingual English plagiarism, whereas the 10% remaining is cross-lingual, from automatic translation from German and Spanish.

5. 50% of the suspicious documents contains plagiarism cases from at least one reference document.

For the experiment purpose, only a small subset from the entire corpus is used. Eighty suspicious documents are chosen from the corpus. Such suspicious documents are known to have plagiarized passages from five hundred sources. These plagiarism cases are only in English. The documents are chosen based on the criterium that the distribution of the plagiarized cases where as is the original corpus. Figure 4.1 illustrate the distribution of source documents plagiarized from different suspicious documents in the case of the original corpus. Figure 4.2 illustrates the same statistics for the small subset chosen. In both cases, there are documents with only one plagiarism case, and others with as much as 30.

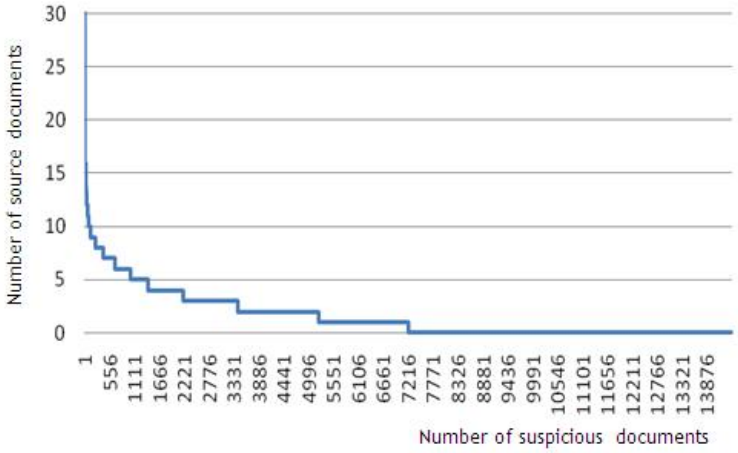

Figure 4.1: Distribution of number of sources plagiarized from suspicious documents, original DB.

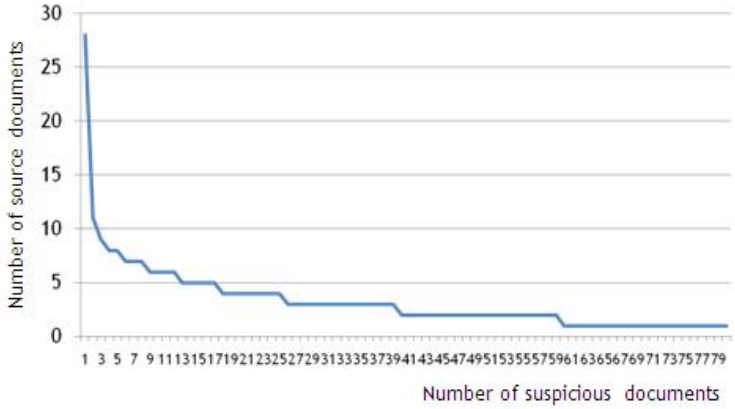

Figure 4.2: Distribution of number of sources plagiarized from suspicious documents, experiment sample.

4.2 Experiment Design

The experiment consists on comparing each suspicious document against the five hundred possible sources. The output of the algorithm and the runtime will be stored. Once all of the suspicious have been verified, the results are going to be compared with those of the annotations given.

By contrasting the obtained results with the annotations the performance measures *Accuracy*, *Recall*, *Precision* and *F-Measure* are going to be computed, along with the run time for each algorithm. Each of the performance measures are detailed in section 4.4.

4.3 Algorithm Parameters and Benchmark Algorithms

The algorithms for experimental purposes are listed in Table 4.1. The parameters used are for the gram structure size, the size of the window for comparing n-grams, the number of coincidence required to increase the similarity indicator. Each of there parameters are configured and used to look for plagiarized passages.

Four algorithms are used for experimental and testing purposes. Three of the selected algorithms are based on the previous approach presented in section 3.2.1 and a variation of the `unix diff` command used to detect changes between two documents was used as benchmark.

The first, named "SimParalell" is an iteration where the pair of documents is compared exhaustively. The parameters used are n the parameter of the gram structure, m is the size of an sliding window to be considered. Parameter K represents the minimum number of common n-grams to increase a counter indicator. Finally, parameter C is the number of cores used in a parallelized implementation of the algorithm.

The second algorithm, "SimTF", is equivalent to algorithm 3.2.2, but the sorting strategy is based on *term frequency*. In this case it is expected a faster running time than "SimParalell", at a cost of a possibly loss of recall because of the approximated nature of the approach. Then, "SimAR" is the algorithm 3.2.2 whose pseudo-code is presented in section 3.2.1. In this case, as well as "SimTF", it is expected a faster running time than "SimParalell" at a cost of a possibly loss of recall.

Finally, the "Diff" approach is a basic algorithm based on the `unix` command `diff`. This approach is based on the move, delete and add characteristics presented by the command, where each one of these outputs is used to determine the scoring function for plagiarism detection.

All of these algorithms outputs are considered as an approximation to the plagiarism detection problem, for which further analysis needs to be taken into consideration for a given pair of documents. They do not offer the offset nor the length of the plagiarism passages, however they

determine how close a pair of documents are.

Table 4.1: Algorithms and their parameters used for the conducted experiment.

Name	Description	Parameters
SimParalell0	CD Sim paralell original	$(n = 3, m = 5, K = 3, c = 16)$
SimParalell1	CD Sim paralell modified 1	$(n = 2, m = 6, K = 3, c = 16)$
SimParalell2	CD Sim paralell modified 2	$n = 4, m = 8, K = 3, c = 16$
SimTF0	CD Sim TF original	$(n = 3, m = 5, \theta_1 = 7, \theta_2 = 2, k = 150)$
SimTF1	CD Sim TF modified 1	$(n = 2, m = 6, \theta_1 = 7, \theta_2 = 2, k = 50)$
SimTF2	CD Sim TF modified 2	$(n = 4, m = 8, \theta_1 = 7, \theta_2 = 2, k = 150)$
SimVP0	CD Sim AR original	$(n = 3, m = 5, \theta_1 = 18, \theta_2 = 5, k = 250)$
SimVP1	CD Sim AR modified 1	$(n = 2, m = 6, \theta_1 = 18\theta_2 = 5, k = 250)$
SimVP2	CD Sim AR modified 2	$(n = 4, m = 8, \theta_1 = 18, \theta_2 = 5, k = 250)$
Diff0	CD Diff original	(Add = -1 , Move = 10 , Delete = -1)
Diff1	CD Diff modified 1	(Add = -10 , Move = 0 , Delete = -10)
Diff2	CD Diff modified 2	(Add = -5 , Move = 0 , Delete = -10)

4.4 Evaluation Criteria

The resulting confusion matrix of this binary classification task can be described by using four possible outcomes: Correctly classified plagiarized documents or True Positives (TP), correctly classified non plagiarized documents or True Negative (TN), wrong classified non plagiarized documents as plagiarized or False Positive (FP), and wrong classified plagiarized documents as non-plagiarized or False Negative (FN).

The evaluation criteria considered are common information retrieval measures, which are constructed using the before mentioned classification outcomes. Also, the runtime for each algorithm is included.

- Precision, that states the degree in which a pair of documents identified as a plagiarism case have indeed copy between them, and Recall, that states the percentage of plagiarized documents that the classifier manages to classify correctly. These measures can be interpreted in conjunction as the classifier's effectiveness. *TP* means "True Positive"; documents found to be plagiarized, detection which is correct. *FP* means that a document that should have been identified as plagiarized, was not. *TN* means that a document was classified as plagiarized, which is incorrect. And finally, *FN* indicates a document that was not identified as plagiarized, when the correct decision would have been the opposite.

$$\text{Precision} = \frac{TP}{TP + FP} \text{ , Recall} = \frac{TP}{TP + FN} \tag{4.1}$$

- F-measure, the harmonic mean between the precision and recall, and Accuracy, the overall percentage of correctly classified documents.

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}, \text{ Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

- Runtime. For each algorithm, the time for performing the 80*500=40.000 pair-document comparison will be annotated.

Chapter 5

Results

The previous algorithms were tested using the evaluation criteria on the selected corpus from the PAN'09 dataset. All results are presented in table 4.3, where the accuracy, precision, recall, F-measure and the runtime are listed. The overall evaluation was performed for each plagiarized case, where for a given suspicious document, the confusion matrix was determined and their performance measures were calculated.

Table 5.1: Results for Accuracy, Precision, Recall, F-measure and runtime for each algorithm presented in section 4.3

Copy Detector	**Accuracy**	**Precision**	**Recall**	**F-measure**	**runtime (s)**
SimParalell0	**0,998**	**0,895**	0,914	**0,904**	20568
SimParalell1	0,990	0,616	0,958	0,750	21103
SimParalell2	0,961	0,882	0,916	0,899	29655
SinTF0	0,874	0,824	0,821	0,823	6959
SinTF1	0,923	0,766	0,800	0,783	7451
SinTF2	0,874	0,836	0,818	0,827	6615
SinAR0	0,887	0,865	0,856	0,861	5393
SinAR1	0,899	0,859	0,852	0,855	5596
SinAR2	0,849	0,828	0,868	0,847	**5231**
Diff0	0,584	0,005	0,348	0,010	6617
Diff1	0,007	0,007	**1,000**	0,013	6529
Diff2	0,584	0,005	0,348	0,010	6179

The results for the experiment are listed in Table 5.1. These results were computed on a notebook computer, consisting of 1.86Ghz Intel CPU and 1GB of RAM.

As the numbers indicate, the best results in term of F-Measure are obtained with "SimParalell". This comes to no surprise, as the algorithm exhaustively checks the documents. For

"SimParalell" the results remain the best even when modifying the parameters.

Acceptable results are obtained with "SimTF"; the F-Measure is close to 0.8 in each case. "SimTF" performs the search taking *samples* of word n-grams in each segment. The criterium for selecting those n-grams is based on the *term frequency*. Therefore, given the considerably reduced time it consumes while maintaining acceptable results, this approach is remarkable.

The main contribution, "SimAR", takes a similar approach than "SimTF", as it only considers *samples* of word n-grams for each segment. The difference is that it considers the alphabetical order to rank the n-grams. In this case, the best results for runtime are obtained; close to "SimTF" and to "Diff", but considerably better than "SimParalell". As for the F-Measure, in each case the algorithm gets close to 0.8 value. This proves to be an excellent result.

Finally, "Diff" algorithm perform the worst. It consumes less time to perform the comparisons, but its basic approach, based exclusively on the *diff* command, does not take into account several aspects for written plagiarism, thus obtaining the worst results of the group, close to 0.11 F-Measure.

It is important to note that the approximations used in "SimTF" and in "SimAR" reduce the runtime in almost four times, compared to the runtime of "SimParalell". Given that the F-Measure in both cases remain acceptable, these proved excellent measures to tackle the dimensionality problem, which increases considerably the runtime when the number of documents to be compared increases substantially.

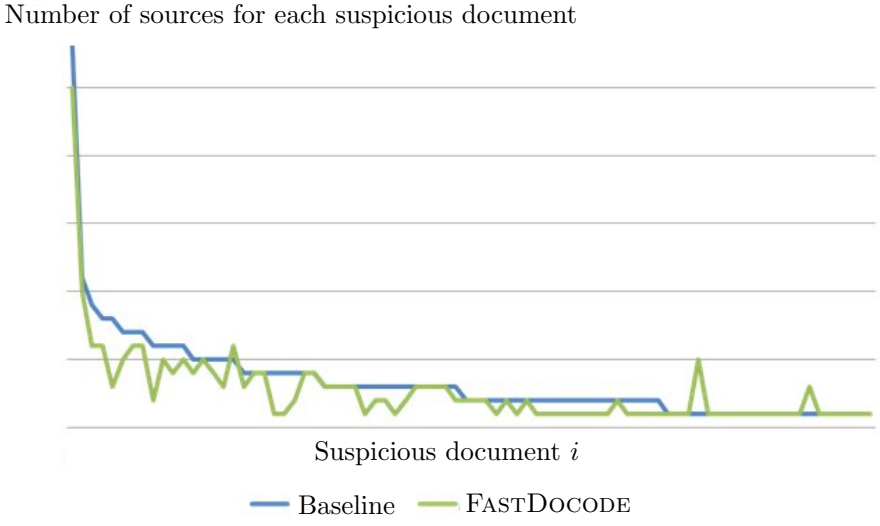

Figure 5.1: Results comparing the baseline sources for suspicious documents (blue line), and those retrieved by FastDocode (green line).

In figure 5.1, results for the SimVP0 algorithm, where the expected curve for source-suspicious relationship is presented together with the source-suspicious relationship that was re-

trieved with the proposed algorithm. These results shows that in the overall evaluation of the selected corpus, the proposal was robust in different number of sources for each suspicious evaluated.

Chapter 6

Conclusions and Future Work

Multiple efforts have been done in the topic of automated plagiarism detection and the main approaches have been described in this thesis. Some focus on intrinsic plagiarism, considering only the suspicious document, whereas other algorithms focus on external plagiarism detection, considering suspicious documents and possible sources.

In this work a method for uncovering external verbatim plagiarism cases have been proposed. The strategy is based on word tri-grams and word bi-grams structures, and consist basically on two phases. The first processes the document leaving it ready for examination, and the second is aimed at approximately search a pair of document for plagiarism signs. This method is therefore designed for comparing lot's of documents.

First, before any document can be processed, it needs to be extracted and cleaned. This step removes any character that is not from the a-z group. Then, all characters are modified to be lowercase. This procedure proved to be important for further analyzing the documents, as it reduces the noise of underlying code, which affects the detection procedure.

Second, to tackle the increasing runtime over size of the documents, this method uses a statistical approach; removes stopwords and the segments to be compared are sampled based on alphabetic order which helps reduce considerably the running time of the algorithm. This proved to be empirically successful but further analysis must be taken into consideration.

Third, all algorithms parameters used were not selected using an extensive analysis on the algorithms performance; due to the size of the corpus it was difficult to run an optimization or grid search strategy over these parameters. However, they were approximated by iteratively experimenting on the sample, thus obtaining acceptable results.

Fourth, although this approach provided acceptable results, more testing is needed, particularly in plagiarism cases where rewording were used, or in other languages as the experiment only

considered English cases.

The implementation of the proposed method will be integrated on a plagiarism detection system, DOCODE. This system will be of help tackling the plagiarism problem in academia, given the increasingly number of cases and the difficulty to check manually for plagiarism each student's work.

It is important to note, that while this method and it's implementation could help in the difficult task of plagiarism uncovering, it is a human that should decide whether or not a document contain plagiarism. Specially at the task of author recognition, this approach is not aimed at uncovering the guilty from the actual author.

Finally, more work is needed, specially at uncovering more complex plagiarism cases, where, for example, rewording is used. Also, as technology could help at plagiarism detection, it could also help to hide it. Trying to use certain software or techniques for avoiding automated plagiarism detection could be a problem and future algorithms should take this into account. This work can also be extended by doing research in the field of multi-lingual plagiarism detection and intrinsic plagiarism detection.

REFERENCES

[1] Jun-Peng Bao, Jun-Yi Shen, Xiao-Dong Liu, Hai-Yan Liu, and Xiao-Di Zhang. Semantic sequence kin: A method of document copy detection. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 529–538. Springer Berlin / Heidelberg, 2004.

[2] Alberto Barrón-Cedeño, Chiara Basile, Mirko Degli Esposti, and Paolo Rosso. Word length n-grams for text re-use detection. In *CICLing '10: Proceedings of the 11th International Conference on Computational Linguistics and Intelligent Text Processing*, pages 687–699, Berlin, Germany, 2010. Springer Berlin / Heidelberg.

[3] Alberto Barrón-Cedeño and Paolo Rosso. Towards the Exploitation of Statistical Language Models for Plagiarism Detection with Reference. In Benno Stein, Efstathios Stamatatos, and Moshe Koppel, editors, *Proceedings of the ECAI'08 PAN Workshop: Uncovering Plagiarism, Authorship and Social Software Misuse*.

[4] Alberto Barrón-Cedeño, Paolo Rosso, and José-Miguel Benedí. Reducing the plagiarism detection search space on the basis of the kullback-leibler distance. In *CICLing '09: Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing*, pages 523–534, Berlin, Heidelberg, 2009. Springer Berlin / Heidelberg.

[5] Chiara Basile, Dario Benedetto, Emanuele Caglioti, Giampaolo Cristadoro, and Mirko Degli Esposti. A plagiarism detection procedure in three steps: Selection, matches and s̈quares. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 19–23. CEUR-WS.org, September 2009.

[6] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

[7] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[8] Felipe Bravo-Marquez, Gastón L'Huillier, Sebastián A. Ríos, Juan D. Velásquez, and Luis A. Guerrero. Docode-lite: A meta-search engine for document similarity retrieval. In R. Setchi

et al., editor, *KES'2010: 14th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 93–102., Berlin Heidelberg, 2010. Springer Berlin / Heidelberg.

[9] Sergey Brin, James Davis, and Héctor García-Molina. Copy detection mechanisms for digital documents. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 398–409, New York, NY, USA, 1995. ACM.

[10] Zdenek Ceska. Plagiarism detection based on singular value decomposition. In *GoTAL '08: Proceedings of the 6th international conference on Advances in Natural Language Processing*, pages 108–119, Berlin, Heidelberg, 2008. Springer Berlin / Heidelberg.

[11] Zdenek Ceska, Michal Toman, and Karel Jezek. Multilingual plagiarism detection. In *AIMSA '08: Proceedings of the 13th international conference on Artificial Intelligence*, pages 83–92, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] Paul Clough. Plagiarism in natural and programming languages: an overview of current tools and technologies. In *Research Memoranda: CS-00-05*, 2000.

[13] Francine Fialkoff. There's no excuse for plagiarism. *Library Journal*, 118(17):56, 1993.

[14] Cristian Grozea, Christian Gehl, and Marius Popescu. Encoplot: Pairwise sequence matching in linear time applied to plagiarism detection. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 10–18. CEUR-WS.org, September 2009.

[15] Patrick Hanks. *Collins Dictionary of the English Language*. Glasgow, William Collins, 1979.

[16] Russell Hunt. Let's hear it for internet plagiarism. *Teaching Learning Bridges*, 2(3):2–5, 2003.

[17] K.O. Jones, J.M.V. Reid, and R Bartlett. Student plagiarism and cheating in an it age. In *Proceedings of the International Conference on Computer Systems and Technology*, pages IV.8:1–6, 2005.

[18] Bao Jun-Peng, Shen Jun-Yi, Liu Xiao-Dong, Liu Hai-Yan, and Zhang Xiao-Di. Document copy detection based on kernel method. In *Proceedings of the 2003 International Conference on Natural Language Processing and Knowledge Engineering.*, pages 250–255, 2003.

[19] NamOh Kang, Alexander Gelbukh, and SangYong Han. Ppchecker: Plagiarism pattern checker in document copy detection. In Petr Sojka, Ivan Kopecek, and Karel Pala, editors, *Text, Speech and Dialogue*, volume 4188 of *Lecture Notes in Computer Science*, pages 661–667. Springer Berlin / Heidelberg, 2006.

[20] Jan Kasprzak, Michal Brandejs, and Miroslav Kripac. Finding plagiarism by evaluating document similarities. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 24–28. CEUR-WS.org, September 2009.

[21] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[22] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.

[23] Caroline Lyon, Ruth Barrett, and James Malcolm. A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector. In *Proceedings of Plagiarism: Prevention, Practice and Policies Conference*, Newcastle, UK, 2004.

[24] Caroline Lyon, James Malcolm, and Bob Dickerson. Detecting short passages of similar text in large document. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 118–125, Pennsylvania, 2001.

[25] Hermann Maurer, Frank Kappe, and Bilal Zaka. Plagiarism - a survey. *Journal of Universal Computer Science*, 12(8):1050–1084, 2006. |http://www.jucs.org/jucs_12_8/plagiarism_a_survey—.

[26] Hermann Maurer and Narayanan Kulathuramaiyer. Coping with the copy-paste-syndrome. In Theo Bastiaens and Saul Carliner, editors, *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2007*, pages 1071–1079, Quebec City, Canada, October 2007. AACE.

[27] Yurii Palkovskii. "counter plagiarism detection software" and "counter counter plagiarism detection" methods. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 67–68. CEUR-WS.org, September 2009.

[28] Chris Park. In other (people's) words: plagiarism by university students – literature and lessons. In *Assessment and Evaluation in Higher Education*, number 5, pages 471–488. Carfax Publishing, 2003.

[29] Martin Potthast, Alberto Barrón-Cedeño, Benno Stein, and Paolo Rosso. Cross-language plagiarism detection. *Language Resources and Evaluation*, pages 1–18, 2010.

[30] Martin Potthast, Benno Stein, Andreas Eiselt, Alberto Barrón-Cedeño, and Paolo Rosso. Overview of the 1st international competition on plagiarism detection. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 1–9. CEUR-WS.org, September 2009.

[31] Bruno Pouliquen, Ralf Steinberger, and Camelia Ignat. Automatic identification of document translations in large multilingual document collections. In *RANLP 2003 - Proceedings of the International Conference on Recent Advances in Natural Language Processing*, pages 401–408, 2003.

[32] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[33] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[34] Leanne Seaward and Stan Matwin. Intrinsic plagiarism detection using complexity analysis. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 56–61. CEUR-WS.org, September 2009.

[35] Jangwon Seo and W. Bruce Croft. Local text reuse detection. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 571–578, New York, NY, USA, 2008. ACM.

[36] Narayanan Shivakumar and Hector Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *DL '96: Proceedings of the first ACM international conference on Digital libraries*, pages 160–168, New York, NY, USA, 1996. ACM.

[37] Antonio Si, Hong Va Leong, and Rynson W. H. Lau. Check: a document plagiarism detection system. In *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 70–77, New York, NY, USA, 1997. ACM.

[38] Efstathios Stamatatos. Intrinsic plagiarism detection using character n-gram profiles. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 38–46. CEUR-WS.org, September 2009.

[39] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer Berlin / Heidelberg, 1999.

[40] Sven Meyer zu Eissen, Benno Stein, and Marion Kulig. Plagiarism detection without reference collections. In Reinhold Decker and Hans-Joachim Lenz, editors, *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 359–366. Springer Berlin / Heidelberg, 2006.

Appendix

Appendix A

Algorithms

The algorithms developed for this work are attached below.

A.1 Approximate Comparison

```
// ALGORITHM FOR COMPARING SEGMENTS OF DOCUMENTS
// TAKES AS INPUTS 2 DOCUMENTS AS STRING ARRAYS
// IT'S OUTPUT INDICATES IF THE PAIR OF DOCUMENTS SHOULD BE FURTHER INVESTIGATED OR
    NOT

import java.util.*;
class APPROXCOMPARISON{

public static double sim(String[] a, String[] b, int n, int tt1, int tt2, int tt3)
    throws Exception{

        //go: size of the sample
        //go2: number of common grams fastcheck needs to validate in order to trigger
             analysis
        //go3: size of the segments
        int go=tt1;
        int go2=tt2;
        int go3=tt3;

        double totalsim=0;

        String[][] aT=partition(a,go3,n);
        String[][] bT=partition(b,go3,n);

        String[][] aP=new String[aT.length][];
        String[][] bP=new String[bT.length][];
```

```
        for(int uu=0;uu<aT.length;uu++) aP[uu]=S2.getbigrams(aT[uu]);
        for(int uu=0;uu<bT.length;uu++) bP[uu]=S2.getbigrams(bT[uu]);

        //sampling: get only the last grams, ranked by alphabetic order
        for(int uu=0;uu<aT.length;uu++) aP[uu]=S2.getlast(aP[uu],go);
        for(int uu=0;uu<bT.length;uu++) bP[uu]=S2.getlast(bP[uu],go);

        //checks every segment-pair using the samples
        //if the check indicates, a more exhaustive analysis is conducted
        for(int e1=0;e1<aT.length;e1++){
                for(int e2=0;e2<bT.length;e2++){
                        if(fastcheck(aP[e1],bP[e2]) && 1==1){
                                totalsim=totalsim+sim(aT[e1],bT[e2]);
                        }
                }
        }
        return totalsim;
}

static public boolean fastcheck(HashSet map,String[]aux2){
        if(map.size()<go2 || aux2.length<go2) return false;
        int count=0;
        for(int i=0;i<aux2.length;i++){
                if(map.contains(aux2[i])) count++;
        }
        if(count>=go2) return true;
        else return false;
}


static public double sim(String[] a, String[] b){
        if(a.length==0 || b.length==0) return 0;
        String[] ngramas1=S2.getbigrams(a);
        String[] ngramas2=S2.getbigrams(b);
        int re=S2.calculatecommons(ngramas1,ngramas2);
        if(re>7) return 1;

        return 0;
}

//used for partitioning a document into segments
static public String[][] partition(String[]a, int go3,int n){
        int ns=(int)a.length/go3; if(ns==0) ns=1;
        String[][] aT=new String[ns][];
        for(int i=0;i<ns;i++){
                int ou=0;
                if(i==0){
                        ou=a.length/ns;
                        aT[i]=new String[ou];
                        for(int r=0;r<aT[i].length;r++) aT[i][r]=a[r];
                }
                else if(i<ns-1){
                        ou=((int)a.length/ns)+n-1;
                        aT[i]=new String[ou];
```

```
                        int au=((int)a.length/ns)*i;
                        for(int r=0;r<aT[i].length;r++) aT[i][r]=a[r+au-n+1];
                }
                else{
                        ou=a.length+n-1-(i)*((int)a.length/ns);
                        aT[i]=new String[ou];
                        int au=i*((int)a.length/ns);
                        for(int r=0;r<aT[i].length;r++) aT[i][r]=a[r+au-n+1];
                }
        }
        return aT;
}
}
```

A.2 In Deep Analysis

```
// ALGORITHM FOR UNCOVERING PLAGIARIZED PASSAGES
// TAKES AS INPUTS 2 DOCUMENTS AS STRING ARRAYS
// IT'S OUTPUT ARE THE FOUND PLAGIARIZED PASSAGES; OFFSETS AND LENGTHS FOR THE TWO
    DOCUMENTS
import java.util.*;
import java.io.*;
class FINDOFFSETLENGTH{

static long ko;
static int go;
static int go2;
static ArrayList marcados1;
static ArrayList marcados2;
static String[]indicesA;
static String[]indicesB;
public static ArrayList similitud(String[] a, String[] b, String nombresource,String
    []indicesAW,String[]indicesBW, int objetivo, ArrayList marc1, ArrayList marc2)
    throws Exception{
        marcados1=marc1;
        marcados1=new ArrayList();
        marcados2=marc2;
        marcados2=new ArrayList();

        indicesA=indicesAW;
        indicesB=indicesBW;

        ArrayList copias4=new ArrayList();

        try{
        String[]ngramas1=S2.obtenergramas(a,2);
        String[]ngramas2=S2.obtenergramas(b,2);
        String[]ngramas13=S2.obtenergramas(a,3);
        String[]ngramas23=S2.obtenergramas(b,3);
        HashSet gramascomunes=S2.reducir66(ngramas13,ngramas23);
```

```
        System.out.println("gramas A: "+ngramas1.length+" gramas B: "+ngramas2.length
            );
        System.out.println(gramascomunes.size());
        HashMap conjunto1=new HashMap();
        for(int i=0;i<ngramas13.length;i++){
                if(gramascomunes.contains(ngramas13[i]) && !conjunto1.containsKey(
                    ngramas13[i])){
                        Integer[]aux=new Integer[1];
                        aux[0]=i;
                        conjunto1.put(ngramas13[i],aux);
                }
                else if(gramascomunes.contains(ngramas13[i]) && conjunto1.containsKey
                    (ngramas13[i])){
                        Integer[]aux=(Integer[])conjunto1.get(ngramas13[i]);
                        Integer[]aux1=new Integer[aux.length+1];
                        for(int rr=0;rr<aux.length;rr++) aux1[rr]=aux[rr];
                        aux1[aux.length]=i;
                        conjunto1.put(ngramas13[i],aux1);
                }
        }
        HashMap conjunto2=new HashMap();
        for(int i=0;i<ngramas23.length;i++){
                if(gramascomunes.contains(ngramas23[i]) && !conjunto2.containsKey(
                    ngramas23[i])){
                        Integer[]aux=new Integer[1];
                        aux[0]=i;
                        conjunto2.put(ngramas23[i],aux);
                }
                else if(gramascomunes.contains(ngramas23[i]) && conjunto2.containsKey
                    (ngramas23[i])){
                        Integer[]aux=(Integer[])conjunto2.get(ngramas23[i]);
                        Integer[]aux1=new Integer[aux.length+1];
                        for(int rr=0;rr<aux.length;rr++) aux1[rr]=aux[rr];
                        aux1[aux.length]=i;
                        conjunto2.put(ngramas23[i],aux1);
                }
        }
        System.out.println(conjunto1.size());
        System.out.println(conjunto2.size());

        ArrayList copias=new ArrayList();


        //-----------------------------------------------------------------------
        //INTENTA PILLAR LAS COPIAS CON MODIFICACIONES
        ArrayList copias2=new ArrayList();
        it1 = conjunto1.keySet().iterator();
        while (it1.hasNext()) {

    String aux1 = (String)it1.next();
        Integer[]indices1=(Integer[])conjunto1.get(aux1);
        Integer[]indices2=(Integer[])conjunto2.get(aux1);
        boolean indicesnomarcados;
        //Itero sobre los indices de la coincidencia.
```

```
for(int i=0;i<indices1.length;i++) for(int j=0;j<indices2.length;j++) {
        int ini1=indices1[i];
        int fin1=indices1[i];
        int ini2=indices2[j];
        int fin2=indices2[j];

        indicesnomarcados=revisarindices(ini1,fin1,marcados1,1);
        if(indicesnomarcados) indicesnomarcados=revisarindices(ini2,fin2,
            marcados2,2);

        //-----------------------------------
        boolean desechar=false;
        if(indicesnomarcados && !desechar){
                int maximo=0; int puntero=0;
                if(ini1-25>=0 && ini2-25>=0 && fin1+25<ngramas1.length &&
                    fin2+25<ngramas2.length){
                int porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-25,fin1+25,
                    ngramas2,ini2-25,fin2+25)*100);
                if(porcentaje1>maximo){ maximo=porcentaje1; puntero=1;
                }
                }
                if(ini1-35>=0 && ini2-25>=0 && fin1+15<ngramas1.length &&
                    fin2+25<ngramas2.length){
                int porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-35,fin1+15,
                    ngramas2,ini2-25,fin2+25)*100);
                if(porcentaje1>maximo){ maximo=porcentaje1; puntero=2;
                }
                }
                if(ini1-15>=0 && ini2-25>=0 && fin1+35<ngramas1.length &&
                    fin2+25<ngramas2.length){
                int porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-15,fin1+35,
                    ngramas2,ini2-25,fin2+25)*100);
                if(porcentaje1>maximo){ maximo=porcentaje1; puntero=3;
                }
                }
                if(puntero==1){
                ini1=ini1-25; fin1=fin1+25;
                ini2=ini2-25; fin2=fin2+25;
                }
                if(puntero==2){
                ini1=ini1-35; fin1=fin1+15;
                ini2=ini2-25; fin2=fin2+25;
                }
                if(puntero==3){
                ini1=ini1-15; fin1=fin1+35;
                ini2=ini2-25; fin2=fin2+25;
                }
                if(puntero==0 || maximo<30) desechar=true;
        }
        indicesnomarcados=revisarindices(ini1,fin1,marcados1,1);
        if(indicesnomarcados) indicesnomarcados=revisarindices(ini2,fin2,
            marcados2,2);

        indicesnomarcados=revisarindices(ini1,fin1,marcados1,1);
```

```
if(indicesnomarcados) indicesnomarcados=revisarindices(ini2,fin2,
    marcados2,2);

//comienza un loop de hacer crecer y ajustar indices...
int ew=0;
while(!desechar && indicesnomarcados && ew<1){
ew++;
int ini1final=ini1;
int fin1final=fin1;
if(indicesnomarcados && !desechar){

while(ini1-1>=0 && ini2-1>=0 && ngramas13[ini1-1].equals(ngramas23[
    ini2-1])
                &&      revisarindices(ini1-1,fin1,marcados1,1) &&
                    revisarindices(ini2-1,fin2,marcados2,2)
                            ){
        ini1--;
        ini2--;
}

while(fin1+1<ngramas13.length && fin2+1<ngramas23.length && ngramas13
    [fin1+1].equals(ngramas23[fin2+1])
                && revisarindices(ini1,fin1+1,marcados1,1) &&
                    revisarindices(ini2,fin2+1,marcados2,2)    ){
        fin1++;
        fin2++;
}

boolean romper=false;
int paso=100;
while(ini1-paso>=0 && ini2-paso>=0 && !romper
&& revisarindices(ini1-paso,fin1,marcados1,1)   && revisarindices(
    ini2-paso,fin2,marcados2,2)){
        String[] grupo1=new String[paso];
        for(int g=0;g<paso;g++) grupo1[g]=ngramas1[ini1-paso+g];
        String[] grupo2=new String[paso];
        for(int g=0;g<paso;g++) grupo2[g]=ngramas2[ini2-paso+g];
        if(S2.acuantoredujo(grupo1,grupo2)>10){
                ini1=ini1-paso;
                ini2=ini2-paso;
        }
        else romper=true;
}
romper=false;
paso=100;
        while(fin1+paso<ngramas13.length && fin2+paso<ngramas23.
            length && !romper //&& indicesnomarcados
                && revisarindices(ini1,fin1+paso,marcados1,1) &&
                        revisarindices(ini2,fin2+paso,marcados2,2)
                    ){
        String[] grupo1=new String[paso];
        for(int g=0;g<paso;g++) grupo1[g]=ngramas1[fin1+g];
        String[] grupo2=new String[paso];
        for(int g=0;g<paso;g++) grupo2[g]=ngramas2[fin2+g];
```

```
        if(S2.acuantoredujo(grupo1,grupo2)>10){
                fin1=fin1+paso;
                fin2=fin2+paso;
        }
        else romper=true;
}
romper=false;
paso=100;
        while(fin1+paso<ngramas13.length && fin2+paso<ngramas23.
            length && !romper //&& indicesnomarcados
                &&      revisarindices(ini1,fin1+paso,marcados1,1) &&
                    revisarindices(ini2,fin2+paso,marcados2,2)){
        int nporcentaje=(int)(S2.porcentaje(ngramas1,ini1,fin1+paso,
            ngramas2,ini2,fin2+paso));
        if(nporcentaje>30){
                fin1=fin1+paso;
                fin2=fin2+paso;
        }
        else romper=true;
}
romper=false;
paso=100;
while(ini1-paso>=0 && ini2-paso>=0 && !romper //&& indicesnomarcados
                &&      revisarindices(ini1-paso,fin1,marcados1,1) &&
                    revisarindices(ini2-paso,fin2,marcados2,2)){
        int nporcentaje=(int)(S2.porcentaje(ngramas1,ini1-paso,fin1,
            ngramas2,ini2-paso,fin2));
        if(nporcentaje>30){
                ini1=ini1-paso;
                ini2=ini2-paso;
        }
        else romper=true;
}
//intentar hacer crecer pero cruzado....

romper=false;
paso=100;
while(ini1-paso>=0 && fin2+paso<ngramas23.length && !romper &&
    indicesnomarcados
                &&      revisarindices(ini1-paso,fin1,marcados1,1) &&
                    revisarindices(ini2,fin2+paso,marcados2,2)){
        int nporcentaje=(int)(S2.porcentaje(ngramas1,ini1-paso,fin1,
            ngramas2,ini2,fin2+paso));
        if(nporcentaje>30 && revisarindices(ini1-paso,fin1,marcados1
            ,1) && revisarindices(ini2,fin2+paso,marcados2,2)){
                ini1=ini1-paso;
                fin2=fin2+paso;
        }
        else romper=true;
}
//intentar hacer crecer pero cruzado....
romper=false;
paso=100;
```

```
while(fin1+paso<ngramas13.length && ini2-paso>=0 && !romper &&
    indicesnomarcados
                &&      revisarindices(ini1,fin1+paso,marcados1,1) &&
                      revisarindices(ini2-paso,fin2,marcados2,2)){
        int nporcentaje=(int)(S2.porcentaje(ngramas1,ini1,fin1+paso,
            ngramas2,ini2-paso,fin2));
        if(nporcentaje>30 && revisarindices(ini1,fin1+paso,marcados1
            ,1) && revisarindices(ini2-paso,fin2,marcados2,2)){
             fin1=fin1+paso;
             ini2=ini2-paso;
        }
        else romper=true;
}
//ajustar inicio y fin
if(indicesnomarcados && 1==0){
int maximo=(int)(S2.porcentaje(ngramas1,ini1,fin1,ngramas2,ini2,fin2)
    *100);
int puntero=0;
if(ini1-25>=0 && ini2-25>=0 && fin1+25<ngramas1.length && fin2+25<
    ngramas2.length
                &&      revisarindices(ini1-25,fin1+25,marcados1,1)
                    && revisarindices(ini2-25,fin2+25,marcados2,2)){
        int porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-25,fin1+25,
            ngramas2,ini2-25,fin2+25)*100);
        if(porcentaje1>maximo){ maximo=porcentaje1; puntero=1;
        }
}
if(ini1-35>=0 && ini2-25>=0 && fin1+15<ngramas1.length && fin2+25<
    ngramas2.length
                &&      revisarindices(ini1-35,fin1+15,marcados1,1)
                    && revisarindices(ini2-35,fin2+15,marcados2,2)){
        int porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-35,fin1+15,
            ngramas2,ini2-25,fin2+25)*100);
        if(porcentaje1>maximo){ maximo=porcentaje1; puntero=2;
        }
}
if(ini1-15>=0 && ini2-25>=0 && fin1+35<ngramas1.length && fin2+25<
    ngramas2.length
            &&      revisarindices(ini1-15,fin1+35,marcados1,1) &&
                revisarindices(ini2-15,fin2+35,marcados2,2)){
         int porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-15,fin1+35,
             ngramas2,ini2-25,fin2+25)*100);
         if(porcentaje1>maximo){ maximo=porcentaje1; puntero=3;
         }
}
if(puntero==1){
        ini1=ini1-25; fin1=fin1+25;
        ini2=ini2-25; fin2=fin2+25;
}
if(puntero==2){
        ini1=ini1-35; fin1=fin1+15;
        ini2=ini2-25; fin2=fin2+25;
}
if(puntero==3){
```

```
                ini1=ini1-15; fin1=fin1+35;
                ini2=ini2-25; fin2=fin2+25;
            }
            if(puntero==0 || maximo<30) desechar=true;
            }
            if(ini1final==ini1 && fin1final==fin1) desechar=true;
            }
            }
            //termina loop de hacer crecer y ajustar indices...
            int porcentaje=(int)(S2.porcentaje(ngramas1,ini1,fin1,ngramas2,ini2,
                fin2)*100);

            indicesnomarcados=revisarindices(ini1,fin1,marcados1,1);
            if(indicesnomarcados) indicesnomarcados=revisarindices(ini2,fin2,
                marcados2,2);

            if(fin1-ini1>50 && indicesnomarcados && porcentaje>10 && !desechar){
            int[]todos=new int[5];
            todos[0]=ini1;
            todos[1]=fin1;
            todos[2]=ini2;
            todos[3]=fin2;
            todos[4]=(int)(S2.porcentaje(ngramas1,ini1,fin1,ngramas2,ini2,fin2)
                *100);
            copias2.add(todos);
            //marcar el tramo de susp y el tramo de source para que no se vuelvan
                 a analizar
            int[]yo=new int[2];
            yo[0]=S2.toInt(indicesA[ini1]);
            yo[1]=S2.toInt(indicesA[fin1]);
            marcados1.add(yo);
            yo=new int[2];
            yo[0]=S2.toInt(indicesB[ini2]);
            yo[1]=S2.toInt(indicesB[fin2]);
            marcados2.add(yo);
    }

    }

}

    //TERMINA PILLAR LAS COPIAS CON MODIFICACIONES
    //-----------------------------------------------------------------
    System.out.println("Size de copias conjuntas: "+copias.size()+" "+copias2.
        size());
    System.out.println("Size de ind marcados: "+marcados1.size()+" "+marcados2.
        size());

    ArrayList copias3=new ArrayList();
    copias3=new ArrayList();
    for(int w=0;w<copias.size();w++) copias3.add(copias.get(w));
    for(int w=0;w<copias2.size();w++){
            int[]todos=(int[])copias2.get(w);
            int porcentaje=todos[4];
```

```
                if(porcentaje>20 ) copias3.add(copias2.get(w));

        }
        copias3=ordenar(copias3,0);

        for(int w=0;w<copias3.size();w++){
                int[]todos=(int[])copias3.get(w);
                int ini1=todos[0]; int fin1=todos[1];
                int ini2=todos[2]; int fin2=todos[3];
                int porcentaje=todos[4];

                int suspoffset=S2.toInt(indicesA[ini1]);
                int susplength=S2.toInt(indicesA[fin1])-S2.toInt(indicesA[ini1]);
                int sourceoffset=S2.toInt(indicesB[ini2]);
                int sourcelength=S2.toInt(indicesB[fin2])-S2.toInt(indicesB[ini2]);
                System.out.println(w+"\t"+porcentaje+"\t Copia: susp_offset: " +
                    suspoffset+"\t susp_length: "+susplength +"  \t source_offset: "
                    +sourceoffset+"\t source_length: "+sourcelength);
        }
        copias3=ordenar(copias3,0);

        for(int w=0;w<copias3.size();w++){
                int[]todos=(int[])copias3.get(w);
                int ini1=todos[0]; int fin1=todos[1];
                int ini2=todos[2]; int fin2=todos[3];
                int porcentaje=todos[4];
                int suspoffset=S2.toInt(indicesA[ini1]);
                int susplength=S2.toInt(indicesA[fin1])-S2.toInt(indicesA[ini1]);
                int sourceoffset=S2.toInt(indicesB[ini2]);
                int sourcelength=S2.toInt(indicesB[fin2])-S2.toInt(indicesB[ini2]);
                Object[] au=new Object[6];
                au[0]=suspoffset;
                au[1]=susplength;
                au[2]=sourceoffset;
                au[3]=sourcelength;
                au[4]=porcentaje;
                au[5]=nombresource;

                copias4.add(au);
        }
        }
        catch(Exception E){System.out.println("error"); E.printStackTrace();}


        return copias4;


}
//funcion que estima el indice del texto sin niuna modificacion
static public int contarchar(int x1, int x2, String[]a){
        int y=0;
        for(int i=x1;i<x2;i++) y=y+a[i].length()+1; //suma el largo de cada palabra
        int r=(int)y/16; //estima los caracteres de puntuacion y demases que podrian
            haberse quitado en la limpieza
```

```
        return y+r;
}
static public boolean revisarindices(int iniBruto, int finBruto, ArrayList marcados,
    int o){
        if(iniBruto<0 || finBruto<0) return false;
        int ini;
        int fin;
        if(o==1){
                ini=S2.toInt(indicesA[iniBruto]);
                fin=S2.toInt(indicesA[finBruto]);
        }
        else{
                ini=S2.toInt(indicesB[iniBruto]);
                fin=S2.toInt(indicesB[finBruto]);
        }
        for(int p=0;p<marcados.size();p++){
                int[] rango=(int[])marcados.get(p);
                int comienzo=rango[0];
                int ultimo=rango[1];
                if(ini>=comienzo && ini<=ultimo) return false;
                else if(fin>=comienzo && fin<=ultimo) return false;
                else if(ini<=comienzo && fin>=ultimo) return false;
                else if(ini>=comienzo && fin<=ultimo) return false;
        }
        return true;
}
//un chekeo de indices marcados para cuando se juntan tramos detectados...
static public boolean rangonousado(int iniBruto, int finBruto, ArrayList marcados,
    int o){
        int ini;
        int fin;
        if(o==1){
                ini=S2.toInt(indicesA[iniBruto]);
                fin=S2.toInt(indicesA[finBruto]);
        }
        else{
                ini=S2.toInt(indicesB[iniBruto]);
                fin=S2.toInt(indicesB[finBruto]);
        }
        if(fin-ini<=5) return true;
        for(int p=0;p<marcados.size();p++){
                int[] rango=(int[])marcados.get(p);
                int comienzo=rango[0];
                int ultimo=rango[1];
                if(ini>=comienzo && fin<=ultimo) return false;
        }
        return true;
}
static public ArrayList ordenar(ArrayList copias, int indice){
        int[] ordenar=new int[copias.size()];
        for(int w=0;w<copias.size();w++){
                int[]todos=(int[])copias.get(w);
                ordenar[w]=todos[indice];
        }
```

```
        Arrays.sort(ordenar);
        //-------------------------------------------------------------
        //INTENTA  JUTAR LOS SEGMENTOS QUE CORRESPONDEN
        ArrayList copias2=new ArrayList();
        for(int v=0;v<ordenar.length;v++){
        for(int w=0;w<copias.size();w++){
        int[] todos=(int[])copias.get(w);
        if(todos[indice]==ordenar[v]){
                int[]yo=new int[5];
                yo[0]=todos[0];
                yo[1]=todos[1];
                yo[2]=todos[2];
                yo[3]=todos[3];
                yo[4]=todos[4];
                copias2.add(yo);
        }}}
        return copias2;
}

static public ArrayList armargrupos(ArrayList copias2,String[]indicesA,String[]
    indicesB){
        copias2=ordenar(copias2,0);
        ArrayList posiblesgrupos=new ArrayList();
        for(int w=0;w<copias2.size();w++){
                ArrayList paraagregar=new ArrayList();
                int[]todos=(int[])copias2.get(w);
                paraagregar.add(todos);
                int ini1=todos[0];
                int fin1=todos[1];
                int ini2=todos[2];
                int fin2=todos[3];
                int ini2provisorio=ini2;
                int fin2provisorio=fin2;
                int porcentaje=todos[4];
                boolean parar=false; int tope=w;
                for(int j=w+1;j<copias2.size() && !parar;j++){
                        int[]todosB=(int[])copias2.get(j);
                        int ini1B=todosB[0];
                        int fin1B=todosB[1];
                        int ini2B=todosB[2];
                        int fin2B=todosB[3];
                        int porcentajeB=todos[4];
                        double diferencia=ini1B-fin1;
                        double diferencia2=ini2-fin2B;

                        if(1==1 && Math.abs(ini1B-fin1)<4000 && rangonousado(fin1,
                            ini1B,marcados1,1) ){
                                fin1=fin1B;
                                paraagregar.add(todosB);
                                w++;
                        }
                        else{
                                parar=true;
                        }
```

```
                }
                posiblesgrupos.add(paraagregar);
        }
        return posiblesgrupos;
}
static public ArrayList resolvergrupos(ArrayList posiblesgrupos, String[] ngramas1,
    String[] ngramas2, String[] indicesA, String[] indicesB){
        ArrayList copias4=new ArrayList();
        for(int w=0;w<posiblesgrupos.size();w++)
    {
        ArrayList grupo=(ArrayList)posiblesgrupos.get(w);
        int ini1final=0;
        int fin1final=0;
        int ini2final=0;
        int fin2final=0;
        if(grupo.size()==1) copias4.add(((int[])grupo.get(0)));
        else{
                int grande=0;
                        for(int r2=0;r2<1;r2++){
                grupo=ordenar(grupo,2);
                grande=grupo.size();
                ArrayList aux=new ArrayList();

                for(int v=0;v<grupo.size()&&1==1;v++){
                        int[] todos=(int[])grupo.get(v);
                        int ini1=todos[0];
                        int fin1=todos[1];
                        int ini2=todos[2];
                        int fin2=todos[3];
                        int ini1provisorio=ini1;
                        int fin1provisorio=fin1;
                        int porcentaje=todos[4];
                        boolean parar=false; int tope=v;
                        for(int j=v+1;j<grupo.size() && !parar&&1==1;j++)
            {
                        int[] todosB=(int[])grupo.get(j);
                        int ini1B=todosB[0];
                        int fin1B=todosB[1];
                        int ini2B=todosB[2];
                        int fin2B=todosB[3];
                        int porcentajeB=todos[4];
                        double diferencia=ini2B-fin2;
                        double diferencia2=ini1B-fin1;
                        if(ini1B<fin1) diferencia2=ini1-fin1B;
                        int largo1=S2.toInt(indicesA[fin1])-S2.toInt(indicesA[ini1]);
                        int largo2=S2.toInt(indicesA[fin1B])-S2.toInt(indicesA[ini1B
                            ]);
                        if(1==1 && Math.abs(fin2-ini2B)<4000  && rangonousado(fin2,
                            ini2B,marcados2,2) && Math.abs(diferencia2)<5000){
                                if(ini1B<ini1provisorio) ini1provisorio=ini1B;
                                if(fin1provisorio<fin1B) fin1provisorio=fin1B;
                                ini2=ini2B;
                                fin2=fin2B;
                                tope++;
```

```
                        }
                        else{
                                parar=true;
                        }
                        }

                        int[]yo=new int[5];
                        todos=(int[])grupo.get(v);
                        yo[2]=todos[2];
                        todos=(int[])grupo.get(tope);
                        yo[3]=todos[3];
                        yo[0]=ini1provisorio; yo[1]=fin1provisorio;

                        yo[4]=(int)(S2.porcentaje(ngramas1,yo[0],yo[1],ngramas2,yo
                            [2],yo[3])*100);
                        aux.add(yo);
                        v=tope;
                }
                grupo=aux;
                }
                for(int v=0;v<grupo.size()&&1==1;v++) copias4.add(grupo.get(v));
                }
        }
        return copias4;
}
//chequea si un tramo chico detectado como copia textual no sea parte de un tramo
    grande con modificaciones
static public boolean revisarbien(int ini1, int fin1, int ini2, int fin2, String[]
    ngramas1, String[]ngramas2){
        //si es de largo suficiente, es improbable que corresponda a parte de una
            copia con modificacion
        if(fin1-ini1>200) return true;
        //ver si es probable que el tramo sea parte de una copia con modificacion...
        int porcentaje1;
        if(ini1-100>=0 && ini2-100>=0){
                porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-100,ini1,ngramas2,ini2
                    -100,ini2)*100);
                if(porcentaje1>5) return false;
        }
        if(fin1+100<ngramas1.length && fin2+100<ngramas2.length){
                porcentaje1=(int)(S2.porcentaje(ngramas1,fin1,fin1+100,ngramas2,fin2,
                    fin2+100)*100);
                if(porcentaje1>5) return false;
        }
        int s=100;
        if(ini1-s>=0 && ini2-s>=0 && fin1+s<ngramas1.length && fin2+s<ngramas2.length
            ){
                porcentaje1=(int)(S2.porcentaje(ngramas1,ini1-s,fin1+s,ngramas2,ini2-
                    s,fin2+s)*100);
                if(porcentaje1>5) return false;
        }
        return true;
}}
```

Appendix B

Research Paper Based On This Work

Derived from this work is a publication, submitted and accepted in Conference on Multilingual and Multimodal Information Access Evaluation, *CLEF* [1], 2010. The paper is attached starting on the next page.

[1] http://clef2010.org/

FASTDOCODE: Finding Approximated Segments of N-Grams for Document Copy Detection

Lab Report for PAN at CLEF 2010

Gabriel Oberreuter[1] and Gaston L'Huillier[1] and Sebastián A. Ríos[1] and Juan D. Velásquez[1]

University of Chile, Department of Industrial Engineering, Santiago, Chile
goberreu@ing.uchile.cl, {glhuilli,srios,jvelasqu}@dii.uchile.cl

Abstract Nowadays, plagiarism has been presented as one of the main distresses that the information technology revolution has lead into our society for which using pattern matching algorithms and intelligent data analysis approaches, these practices could be identified. Furthermore, a fast document copy detection algorithm could be used in large scale applications for plagiarism detection in academia, scientific research, patents, knowledge management, among others. Notwithstanding the fact that plagiarism detection has been tackled by exhaustive comparison of source and suspicious documents, approximated algorithms could lead to interesting results. In this paper, an approach for plagiarism detection is presented. Results in a learning dataset of plagiarized documents from the PAN'09, and its further evaluation in the PAN'10 plagiarism detection challenge, showed that the trade-off between speed and performance could improve other plagiarism detection algorithms.

1 Introduction

Plagiarism in academia is rising and multiple authors have worked to describe this phenomena [11, 12, 20]. As commented by Hunt in [11], "Internet Plagiarism" is referred sometimes as a cataclysmic consequence of the "Information Technology revolution", as it proves to be a big problem in academia. In [20], plagiarism is analyzed from various perspectives and considered as a problem that is growing bigger over time. In [12], the author analyzes different statistical data and the implications of the "IT age".

To tackle this problem, one approach is to try to detect plagiarism. Different methods involving computer aided plagiarism detection have been under research [6, 8, 10, 14, 16, 24, 25], from which different system for automatic plagiarism detection have been developed. However, different ways for neutralizing such detection systems have been presented. Such methods usually involve modifying the text in such way that the presentation of the document remains the same, but the underlying code is different and normally this differentiation render the detection systems useless [19].

Plagiarism detection for document sources can be classified into several categories [7]. From exact document copy, to paraphrasing, different levels of plagiarism techniques can been used in several contexts [14, 28]. Likewise, pairs of documents can be described into different categories as unrelated, related, partly overlapped, subset, and copied.

When comparing a suspicious document against a collection of possible sources, it is tried to identify the sentences, paragraph or ideas that have been copied. This is called external plagiarism detection [22], and multiple efforts are being oriented in this area. Another approach, is to determine within features extracted from just one given document. However, this work is mainly focused on external plagiarism detection, without considering elements from the intrinsic plagiarism detection case.

The main contribution of this work is a technique for plagiarism detection based on a two step evaluation. First, a filter evaluation which considers a fast generation of segments of n-grams for an approximated decision. And second, an obfuscation and exhaustive search process for the offset and length of the plagiarized extraction between two previously classified documents is performed. This two-step algorithm is based on different document pre-processing strategies and decision thresholds which gives a large number of parameters or degrees of freedom to be determined.

This paper is structured as follows: In Section 2 an overview of plagiarism detection algorithms and related work is presented. Then, in Section 3 the proposed FAST Document Copy Detection (FASTDOCODE) method is introduced. Afterwards, in Section 4, the experimental setup and evaluation performance criteria are described. In Section 5 results are discussed. Finally, in Section 6 the main conclusions are presented.

2 Related Work

According to Schleimer et al. [23], copy prevention and detection methods can be combined to reduce plagiarism. While copy detection methods can only minimize it, prevention methods can fully eliminate it and decrease it. Notwithstanding this fact, prevention methods need the whole society to take part, thus its solution is non trivial. Copy or plagiarism detection methods tackle different levels, from simple manual comparison to complex automatic algorithms [22]. Among these techniques, document similarity detection, writing style detection, document content similarity, content translation, multi-site source plagiarism, and multi-lingual plagiarism detection methods have been previously proposed [6, 7, 14, 18, 22, 23, 26].

2.1 Intrinsic Plagiarism Detection

When comparing texts against a reference set of possible sources, comes the complication of choosing the right set. And now more than ever, with the possibilities the internet bring to plagiarists, this task becomes more complicated to achieve. For this, intrinsic plagiarism detection algorithms have been developed [28].

The writer style can be analyzed within the document and an examination for incongruities can be done. The complexity and style of each text can be analyzed based on certain parameters such as text statistics, syntactic features, part-of-speech features, closed-class word sets, and structural features [28]. Whose main idea is to define a criterium to determine if the style has changed enough to indicate plagiarism.

It is important to note that using intrinsic plagiarism detection for both, automated and manual, it is not demonstrated that a paragraph or a part of the document is being copied, because there is no reference to compare to. Therefore this kind of plagiarism detection category is only indicative and should be used in conjunction with human supervision. Nevertheless, intrinsic plagiarism is useful when trying to discover originality and authorship of a document.

2.2 External Plagiarism

Before the comparison between each possible source and the suspicious document can be executed, an important obstacle is to be resolved. This task consist in defining and gathering the possible sources, and this is becoming more and more complex as the technology becomes more available. In [5], the suspicious document is chopped into queries and web search engines are used to obtain a set of candidates sources. This approach helps tackle this problem but more work is needed.

Another issue to be considered, is when the collection of possible sources become too large. The size of the set of possible sources can be thousands of documents. In [22], PAN Competition and Workshop, the external part of the competition, and now merged with the training corpus, considers a set of sources of 14,428 documents or possible sources. In this case solutions do exist, as reducing the search space using different data mining techniques.

In [17], the use of n-grams for plagiarism detection is explored. The use of n-grams gives some flexibility to the detection, as reworded fragments could still be detected. In particular, in [3] the tri-gram structure is found to be the most effective in this task. This method is possible because the common n-grams between two documents are usually a low percentage of the total number of n-grams of both text. Due to this, n-grams could probe promissory for plagiarism detection techniques. Furthremore, in [16], Lyon et al. extended their work and the Ferret system was implemented, which uses this approximation to detect plagiarism. A distance is calculated between the documents, based on the n-grams found in common. The results indicate that this structure is useful and provides flexibility at detecting plagiarism with modifications of words.

Other approaches focus on solving the plagiarism detection problem as a traditional classification problem from the machine learning community [1, 9, 13]. Bao et al. in [13] and then in [1], proposed to use a Semantic Sequence Kernel (SSK), and then using it into a traditional Support Vector Machines (SVMs) formulation based on the Structural Risk Minimization (SRM) [4, 27] principle from statistical learning theory, where the general objective is finding out the optimal classification hyperplane for the binary classification problem (plagiarized, not plagiarized). Likewise, other approaches solves the same classification problem by using Self Organizing Feature Maps (SOFM) [15], with promising results in the classification performance.

An interesting issue is the multi-lingual and cross-language detection of plagiarism. This topic is currently under research [2, 21], where promising results for plagiarism and cross-lingual information retrieval have been presented.

2.3 Reducing Search Space

One of the issues to be resolved in external plagiarism analysis and detection is the number of source document candidates. When the task is to detect plagiarism between a small set of suspicious against a small set of source documents, it is simple to search for plagiarism in every pair of documents. The problem is presented when the universe of possible sources is not well defined, or the set of documents is too large. In this case the approach need to be modified, and those changes usually consists in adding a step in the process of plagiarism discovery: the search space reduction.

The aim of this step is to effectively and efficiently identify which texts are possible sources of plagiarism, if any. Usually multiple statistical tools are used in order to reduce the computational time required for computing a large corpus of documents while trying to maintain accuracy at determining which sources need to be discarded.

3 Proposed Method

In this section, the main contribution of our work is described. In the first place, the overall FASTDOCODE algorithm is presented in terms of previously introduced notation. Then, the two steps that defines FASTDOCODE, that is, the approximated segment finding algorithm and the exhaustive offset and length search, are presented in subsections 3.2 and 3.3 respectively. In this section all algorithms are presented as pseudocode, together with a brief description on how different parameters could be used.

Let us introduce some concepts. In the following, let $\mathcal{V}$ a vector of words that defines the vocabulary to be used. We will refer to a word w, as a basic unit of discrete data, indexed by $\{1, ..., |\mathcal{V}|\}$. A document d is a sequence of S words ($|d| = S$) defined by $\mathbf{w} = (w^1, ..., w^S)$, where w^s represents the s^{th} word in the message. Finally, a corpus is defined by a collection of $\mathcal{D}$ documents denoted by $\mathcal{C} = (\mathbf{w}_1, ..., \mathbf{w}_{|\mathcal{D}|})$.

3.1 FASTDOCODE

Given a wide set of parameters $\mathcal{D}_{\text{source}}, \mathcal{D}_{\text{suspicious}}, n, k, m,$ SORTSTRATEGY$, \theta_1, \theta_2, \tau_{min}, St, Pe$, the algorithm tries to find for a corpus $\mathcal{C} = \{\mathcal{D}_{\text{source}}, \mathcal{D}_{\text{suspicious}}\}$ all plagiarized documents in the suspicious partition, using as search space the source partition. This algorithm is based on external plagiarism detection, and does not include intrinsic plagiarism nor multi-lingual evaluation. In general terms, the algorithm first reduces the search space by using an approximated search of segments of n-grams, and then within selected pairs of documents, using an exhaustive search algorithm, finds the offset and its length for both exact and obfuscated copy.

Algorithm 3.1: FAST-DOCODE

Data: $\mathcal{D}_{\text{source}}, \mathcal{D}_{\text{suspicious}}, n, k, m,$ SORTSTRATEGY$, \theta_1, \theta_2, \tau_{min}, St, Pe$
Result: Vector OL with all Offsets and their lengths for the complete corpus of documents

Initialize Vector $pair \leftarrow \{\}$ and $Offset Lenght \leftarrow \{\}$;
foreach $\mathbf{d}_i \in \mathcal{D}_{suspicious}$ **do**
  $(\kappa_i, \mathbf{t}_i) \leftarrow$ PREPROCESSDOCUMENT$(\mathbf{d}_i, n = 3, k, m)$;
  foreach $\mathbf{d}_j \in \mathcal{D}_{source}$ **do**
    $(\kappa_j, \mathbf{t}_j) \leftarrow$ PREPROCESSDOCUMENT$(\mathbf{d}_j, n = 3, k, m)$;
    if APPROXIMATECOMPARISON$(\kappa_i, \kappa_j, \mathbf{t}_i, \mathbf{t}_j, \theta_1, \theta_2)$ **then**
      $p(i, j) \leftarrow (\mathbf{d}_i, \mathbf{d}_j)$;
      $pair$.add(p);
foreach $p \in pair$ **do**
  $\mathbf{t}_i \leftarrow$ PREPROCESSDOCUMENT$(p.\mathbf{d}_i, n = 2, k, m,$ SORTSTRATEGY$)$;
  $\mathbf{t}_j \leftarrow$ PREPROCESSDOCUMENT$(p.\mathbf{d}_j, n = 2, k, m,$ SORTSTRATEGY$)$;
  OL.add(FINDOFFSETLENGHT1$(\mathbf{t}_i, \mathbf{t}_j, \tau_{min}, St, Pe)$);
  OL.add(FINDOFFSETLENGHT2$(\mathbf{t}_i, \mathbf{t}_j, \tau_{min}, St, Pe)$);
return OL ;

In algorithm 3.1, the general evaluation of a corpus is presented. In particular, different procedures are used within the code which helps in the preprocessing of documents. The method PREPROCESSDOCUMENT, presented in algorithm 3.2, takes as input a given document, and returns a set of n-grams or segments of n-grams given the case. If $n = 2$, only a set of bi-grams will be computed, and if $n = 3$ a process of finding segments of n-grams will be performed. Segments of n-grams will be intensively used in the approximated search for reducing the search space, whether the bi-grams will be used in finding all offsets and their lengths.

Algorithm 3.2: PREPROCESSDOCUMENT

```
Data: d_i, n, k, m
if n = 3 then
    RemoveStopWords(d_i);
    t_i ← GenerateNGrams(d_i, n) ;
    k_i ← GenerateKNGrams(t_i, k);
    k_i* ← Sort(k_i, SortStrategy) ;
    κ_i ← SelectMLastNGrams(k_i*, m) ;
    return (κ_i, t_i);
else
    t_i ← GenerateNGrams(d_i, n) ;
    return t_i;
```

As presented in algorithm 3.2, new methods are introduced for the processing, such as the GENERATENGRAMS function that takes a given document d_i and returns a set of n-grams with the structure $(w_i, w_{i+1} \ldots, w_{i+n}), \forall i \geq 1, n \leq S$. Function GENERATEKNGRAMS, generates groups of length k using all n-grams. Then, a SORT algorithm is used within segments, with a specific sorting strategy. In this research, an alphabet sorting strategy and a Term Frequency sorting strategy where used as a variation on the proposed algorithm. Finally, a SELECTMLASTNGRAMS function, as specified in its name definition, selects only the last m n-grams within the segment. This approach can be considered as an analogy to a sampling strategy for each segment, thus contributing to minimize the number of comparisons to be executed and enhancing the runtime of the algorithm.

3.2 Finding Segments Approximation

Algorithm 3.3: APPROXIMATECOMPARISON

```
Data: κ_i, κ_j, t_i, t_j, θ_1, θ_2
if sMatch(t_i, t_j, s ≥ 1) then
    if sMatch(κ_i, κ_j, s ≥ θ_1) then
        if sMatch(t_i, t_j, s ≥ θ_2) then
            return true ;
        end
    end
end
else
    return false;
end
```

Once documents d_i and d_j are processed in n-grams and segments of n-grams, t_i, t_j and κ_i, κ_j respectively, a set of conditions are evaluated in order to set the relation that document d_i has with document d_j, that is, if they are somehow related (algorithm 3.3 returns true), or if it is not worthy to keep finding further relationships (algorithm 3.3 returns false). In this sense, this is an approximated finding procedure that considers both n-grams and their k segments to decide if there is enough information to classify as plagiarism or not.

Algorithm 3.3 first evaluates an $\text{SMATCH}(\mathbf{t}_i, \mathbf{t}_j, s \geq 1)$ algorithm which returns true whether at least one n-gram from $\mathbf{t}_i$ matches one n-gram from $\mathbf{t}_j$. Also a variation of previous matching method is used within the segments of n-grams. Condition $\text{SMATCH}(\kappa_i, \kappa_j, s \geq \theta_1)$ states that at least θ_1 n-grams must

match in between segments κ_i and κ_j. If this is hold, the next condition $\text{SMATCH}(\mathbf{t}_i, \mathbf{t}_j, s \geq \theta_2)$ is associated to find whether at least θ_2 n-grams matches between $\mathbf{t}_i$ and $\mathbf{t}_j$. In general terms, this procedure helps on reducing the search space, and improving the algorithm in both execution time and hardware requirements. By using these constraints, it is possible now to go into a further algorithm for finding the needed offset and its length.

3.3 Find the Offset and its Length

Algorithm 3.4 describes roughly how to find the offset and its length within two documents.

Algorithm 3.4: FINDOFFSETLENGTH

Data: $\mathbf{d}_i, \mathbf{d}_j, \mathbf{t}_i, \mathbf{t}_j, \tau_{min}, St, Pe$
// Find obfuscated and textual copy within documents d_i and d_j
foreach $w_i \in \mathbf{d}_i$ **do**
  foreach $w_j \in \mathbf{d}_j$ **do**
    Initialize Vector $bp(i)^{\text{left}} \leftarrow \{\}$, $bp(j)^{\text{right}} \leftarrow \{\}$, $bp(j)^{\text{left}} \leftarrow \{\}$, $bp(i)^{\text{right}} \leftarrow \{\}$;
    // Move the XMATCH in both $\leftarrow$ and $\rightarrow$ sides of the document in steps St and checking that Pe percentage of similar words within the step
    repeat
      $bp(i)^{\text{right}}.add(t_i)$ and $bp(j)^{\text{right}}.add(t_j)$
    until $!\text{XMATCH}(t_i, t_j, s = 1, \leftarrow, St, Pe)$;
    repeat
      $bp(i)^{\text{left}}.add(t_i)$ and $bp(j)^{\text{left}}.add(t_j)$
    until $!\text{XMATCH}(t_i, t_j, s = 1, \rightarrow, St, Pe)$;
    if $max\{|bp(i)^{left} - bp(i)^{right}|, |bp(j)^{left} - bp(j)^{right}|\} > \tau_{min}$ **then**
      $Offsetlength(i).\text{add}(bp(i)^{\text{left}}, |bp(i)^{\text{left}} - bp(i)^{\text{right}}|)$;
      $Offsetlength(j).\text{add}(bp(j)^{\text{left}}, |bp(j)^{\text{left}} - bp(j)^{\text{right}}|)$;
    end
    // Remove all words inside break points for both d_i and d_j
    REMOVEINCLUDEDWORDS$(\mathbf{t}_i, bp(i)^{\text{left}}, bp(i)^{\text{right}})$;
    REMOVEINCLUDEDWORDS$(\mathbf{t}_j, bp(j)^{\text{left}}, bp(j)^{\text{right}})$;
    UPDATE$(\mathbf{d}_i)$;
    UPDATE$(\mathbf{d}_j)$;
  end
end
return $Offsetlenght$;

Previous algorithm 3.4, finds obfuscated and textual copy within documents d_i and d_j, then a match strategy is moved both left and right side of the document, adding to the offset array the matching segments. Finally, to avoid the search over detected plagiarism passages, the break points are saved and used to remove them.

4 Experiments

In this section, the experimental setup and the evaluation criteria is presented. First, the selected partition of a plagiarism detection corpus from the PAN'09 [22] is discussed together with some of the parameters selected to evaluate different benchmark plagiarism detection algorithms. Then, the evaluation criteria and performance measures used for the training step of the algorithm are presented.

4.1 Experimental Setup

The PAN'09 plagiarism detection corpus [22] was used as a seed to train different plagiarism detection algorithms.

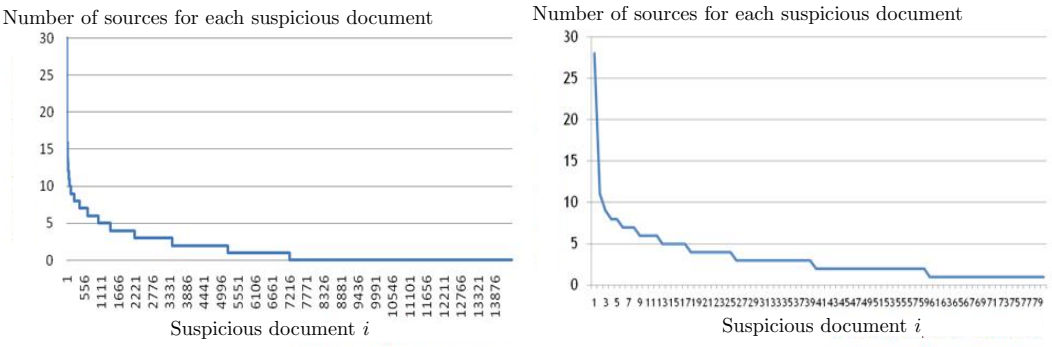

Figure 1. Dataset distribution in terms of the construction of the number of source documents that each suspicious document was originated from.

For the experiment, a small sample of the PAN'09 corpus is chosen. This sample considers only external English plagiarism cases. It is constructed as Figure 1 suggests: maintaining the number of references per suspicious document from the original corpus.

Four algorithms are used for experimental and testing purposes. Three of the selected algorithms are based on the previous approach presented in section 3 and a variation of the `unix diff` command used to detect changes between two documents was used as benchmark. Due to the lack of space, only a brief description of each of the selected algorithms is presented. Further information was intentionally discarded by authors.

The first, named "SimParalell" is an iteration where the pair of documents is compared exhaustively. The parameters used are n the parameter of the gram structure, sw is the size of an n-gram sliding window to be considered. Parameter K represents the minimum number of common n-grams to increase a counter indicator. Finally, parameter C is the number of cores used in a parallelized implementation of the algorithm.

The second algorithm, "SimTF", is equivalent to algorithm 3.3, but the sorting strategy is based on *term frequency*. In this case it is expected a faster running time than the latter, at a cost of a possibly loss of recall because of the approximated nature of the approach. Then, "SimVP" is the algorithm 3.3 whose pseudo-code is presented in section 3. In this case, as well as "SimTF", it is expected a faster running time than "SimParalell" at a cost of a possibly loss of recall.

Finally, the "Diff" approach is a basic algorithm based on the `unix` command `diff`. This approach is based on the move, delete and add characteristics presented by the command, where each one of these outputs is used to determine the scoring function for plagiarism detection.

All of these algorithms outputs are considered as an approximation of the plagiarism detection problem, for which further analysis needs to be taken into consideration for a given pair of documents. They do not offer the offset nor the length of the plagiarism passages, however they determine how close a pair of documents are.

4.2 Evaluation Criteria

The resulting confusion matrix of this binary classification task can be described using four possible outcomes: Correctly classified plagiarized documents or True Positives (TP), correctly classified non plagiarized documents or True Negative (TN), wrong classified non plagiarized documents as plagiarized or False Positive (FP), and wrong classified plagiarized documents as non-plagiarized or False Negative (FN).

The evaluation criteria considered are common information retrieval measures, which are constructed using the before mentioned classification outcomes.

– Precision, that states the degree in which a pair of documents identified as a plagiarism case have indeed copy between them, and Recall, that states the percentage of plagiarized documents that the classifier manages to classify correctly. Can be interpreted as the classifier's effectiveness.

$$\text{Precision} = \frac{TP}{TP + FP}\ , \text{Recall} = \frac{TP}{TP + FN} \tag{1}$$

Table 1. Algorithms and their parameters used for the conducted experiment.

Name	Description	Parameters
SimParalell0	CD Sim paralell original	$(n = 3, sw = 5, K = 3, c = 16)$
SimParalell1	CD Sim paralell modified 1	$(n = 2, sw = 6, K = 3, c = 16)$
SimParalell2	CD Sim paralell modified 2	$n = 4, sw = 8, K = 3, c = 16$
SimTF0	CD Sim TF original	$(n = 3, sw = 5, \theta_1 = 7, \theta_2 = 2, k = 150)$
SimTF1	CD Sim TF modified 1	$(n = 2, sw = 6, \theta_1 = 7, \theta_2 = 2, k = 50)$
SimTF2	CD Sim TF modified 2	$(n = 4, sw = 8, \theta_1 = 7, \theta_2 = 2, k = 150)$
SimVP0	CD Sim AR original	$(n = 3, sw = 5, \theta_1 = 18, \theta_2 = 5, k = 250)$
SimVP1	CD Sim AR modified 1	$(n = 2, sw = 6, \theta_1 = 18\theta_2 = 5, k = 250)$
SimVP2	CD Sim AR modified 2	$(n = 4, sw = 8, \theta_1 = 18, \theta_2 = 5, k = 250)$
Diff0	CD Diff original	(Add = -1 , Move = 10 , Delete = -1)
Diff1	CD Diff modified 1	(Add = -10 , Move = 0 , Delete = -10)
Diff2	CD Diff modified 2	(Add = -5 , Move = 0 , Delete = -10)

– F-measure, the harmonic mean between the precision and recall, and Accuracy, the overall percentage of correct classified documents.

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}, \text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

5 Results and Discussions

Previous algorithms were evaluated using the evaluation criteria on the selected corpus from the PAN'09 dataset. All results are presented in table 2, where the accuracy, precision, recall, F-measure and the evaluation runtime are listed. The overall evaluation was performed for each plagiarized case, where for a given suspicious document, the confusion matrix was determined and their performance measures were evaluated. Then, after all suspicious documents were evaluated, the mean performance was recorded and listed in table 2.

Table 2. Results for Accuracy, Precision, Recall, F-measure and runtime for each algorithm presented in section 4

Copy Detector	Accuracy	Precision	Recall	F-measure	runtime (s)
SimParalell0	**0.999**	0.895	0.914	**0.904**	20,568
SimParalell1	0.990	0.616	0.958	0.750	21103
SimParalell2	0.961	**0.882**	0.916	0.899	29,655
SimTF0	0.874	0.824	0.821	0.823	6,959
SimTF1	0.923	0.766	0.800	0.783	7,451
SimTF2	0.874	0.836	0.818	0.827	6,615
SimVP0	0.887	0.865	0.857	0.861	5,393
SimVP1	0.899	0.859	0.852	0.855	5,596
SimVP2	0.849	0.828	0.868	0.847	**5,231**
Diff0	0.584	0.005	0.349	0.010	6,617
Diff1	0.007	0.007	**1.000**	0.014	6,529
Diff2	0.584	0.005	0.349	0.010	6,179

The results for the experiment are listed in Table 2. As the numbers indicate, the best results in term of F-measure are obtained with "SimParalell". This comes to no surprise, as the algorithm exhaustively checks the documents. The cost of such results, however, is the worst running time of the group. Alternatively, the "SimTF" and "SimVP" both get acceptable results but much better running times than "SimParalell". This factor is important as the number of pair of documents to compare becomes increasingly high. The "Diff" variant gets an overall worse result; based on the `diff` unix command entirely this approach does not take into account different obfuscation levels.

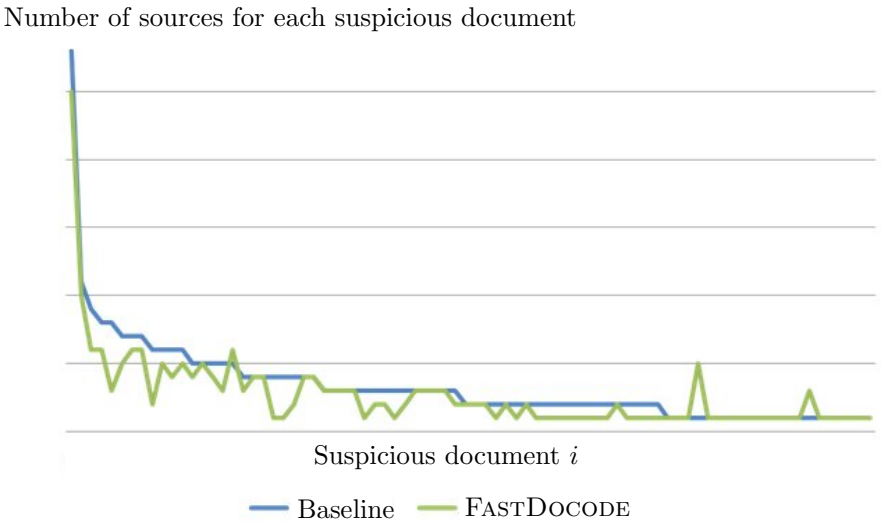

Figure 2. Results comparing the baseline sources for suspicious documents (blue line), and those retrieved by FastDocode (green line).

In Figure 2, results for the SimVP0 algorithm, where the expected curve for source-suspicious relationship is presented together with the source-suspicious relationship that was retrieved with the proposed algorithm. These results show that in the overall evaluation of the selected corpus, our proposal was robust in different number of sources for each suspicious evaluated.

6 Conclusions

In this work we have presented a method for uncovering external plagiarism cases. The strategy proposed is based on word tri-grams and word bi-grams, and consists basically on two phases. The first is aimed at reducing the search space for possible sources, and the second is aimed at exhaustively search a pair of document for plagiarized passages, where the offset and its length are computed.

While reducing the search space, we proposed a method that uses a statistical approach; removing stopwords and selecting samples based on alphabetic order, which helps to reduce considerably the running time of the algorithm. This proved to be empirically successful but further analysis must be taken into consideration.

Second, all algorithms parameters used were not selected using an extensive analysis on the algorithms performance; due to the size of the corpus it was difficult to run an optimization or grid search strategy over these parameters. We did, however, approximate them by iterating and trying on our sample, thus obtaining acceptable results.

As future work, it could be interesting to experiment the proposed approach with char n-grams instead of word n-grams. This could help FastDOCODE to include an intrinsic evaluation of a given document, or help the algorithm to detect plagiarized passages with high obfuscation levels.

7 Acknowledgment

Authors would like to thank continuous support of “Instituto Sistemas Complejos de Ingeniería” (ICM: P-05-004- F, CONICYT: FBO16; www.sistemasdeingenieria.cl); FONDEF project (DO8I-1015) entitled, DOCODE: Document Copy Detection (www.docode.cl); and the Web Intelligence Research Group (wi.dii.uchile.cl).

References

[1] Jun-Peng Bao, Jun-Yi Shen, Xiao-Dong Liu, Hai-Yan Liu, and Xiao-Di Zhang. Semantic sequence kin: A method of document copy detection. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 529–538. Springer Berlin / Heidelberg, 2004.

[2] Alberto Barrón-Cedeño. On the mono- and cross-language detection of text reuse and plagiarism. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 914–914, New York, NY, USA, 2010. ACM.

[3] Alberto Barrón-Cedeño, Chiara Basile, Mirko Degli Esposti, and Paolo Rosso. Word length n-grams for text re-use detection. In *CICLing '10: Proceedings of the 11th International Conference on Computational Linguistics and Intelligent Text Processing*, pages 687–699, Berlin, Germany, 2010. Springer Berlin / Heidelberg.

[4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[5] Felipe Bravo-Marquez, Gastón L'Huillier, Sebastián A. Ríos, Juan D. Velásquez, and Luis A. Guerrero. Docodelite: A meta-search engine for document similarity retrieval. In R. Setchi et al., editor, *KES'2010: 14th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 93–102., Berlin Heidelberg, 2010. Springer Berlin / Heidelberg.

[6] Sergey Brin, James Davis, and Héctor García-Molina. Copy detection mechanisms for digital documents. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 398–409, New York, NY, USA, 1995. ACM.

[7] Zdenek Ceska. The future of copy detection techniques. In *Proceedings of the 1st Young Researchers Conference on Applied Sciences (YRCAS 2007)*, pages 5–107, Pilsen, Czech Republic, November 2007.

[8] Zdenek Ceska. Plagiarism detection based on singular value decomposition. In *GoTAL '08: Proceedings of the 6th international conference on Advances in Natural Language Processing*, pages 108–119, Berlin, Heidelberg, 2008. Springer Berlin / Heidelberg.

[9] Tommy W. S. Chow and M. K. M. Rahman. Multilayer som with tree-structured data for efficient document retrieval and plagiarism detection. *Trans. Neur. Netw.*, 20(9):1385–1402, 2009.

[10] C. Grozea, C. Gehl, and M. Popescu. Encoplot: Pairwise sequence matching in linear time applied to plagiarism detection. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 10–18. CEUR-WS.org, September 2009.

[11] Russell Hunt. Let's hear it for internet plagiarism. *Teaching Learning Bridges*, 2(3):2–5, 2003.

[12] K.O. Jones, J.M.V. Reid, and R Bartlett. Student plagiarism and cheating in an it age. In *Proceedings of the International Conference on Computer Systems and Technology*, pages IV.8:1–6, 2005.

[13] Bao Jun-Peng, Shen Jun-Yi, Liu Xiao-Dong, Liu Hai-Yan, and Zhang Xiao-Di. Document copy detection based on kernel method. In *Proceedings of the 2003 International Conference on Natural Language Processing and Knowledge Engineering.*, pages 250–255, 2003.

[14] NamOh Kang, Alexander Gelbukh, and SangYong Han. Ppchecker: Plagiarism pattern checker in document copy detection. In Petr Sojka, Ivan Kopecek, and Karel Pala, editors, *Text, Speech and Dialogue*, volume 4188 of *Lecture Notes in Computer Science*, pages 661–667. Springer Berlin / Heidelberg, 2006.

[15] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[16] Caroline Lyon, Ruth Barrett, and James Malcolm. A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector. In *Proceedings of Plagiarism: Prevention, Practice and Policies Conference*, Newcastle, UK, 2004.

[17] Caroline Lyon, James Malcolm, and Bob Dickerson. Detecting short passages of similar text in large document. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 118–125, Pennsylvania, 2001.

[18] H. Maurer, F. Kappe, and B. Zaka. Plagiarism - a survey. *Journal of Universal Computer Science*, 12(8):1050–1084, 2006. | http://www.jucs.org/jucs_12_8/plagiarism_a_surveyl.

[19] Yurii Palkovskii. "counter plagiarism detection software" and "counter counter plagiarism detection" methods. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 67–68. CEUR-WS.org, September 2009.

[20] C Park. In other (people's) words: plagiarism by university students – literature and lessons. In *Assessment and Evaluation in Higher Education*, number 5, pages 471–488. Carfax Publishing, 2003.

[21] Martin Potthast, Alberto Barrón-Cedeño, Benno Stein, and Paolo Rosso. Cross-language plagiarism detection. *Language Resources and Evaluation*, pages 1–18, 2010.

[22] Martin Potthast, Benno Stein, Andreas Eiselt, Alberto Barrón-Cedeño, and Paolo Rosso. Overview of the 1st international competition on plagiarism detection. In Benno Stein, Paolo Rosso, Efstathios Stamatatos, Moshe Koppel, and Eneko Agirre, editors, *SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09)*, pages 1–9. CEUR-WS.org, September 2009.

[23] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, New York, NY, USA, 2003. ACM.

[24] Narayanan Shivakumar and Hector Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *DL '96: Proceedings of the first ACM international conference on Digital libraries*, pages 160–168, New York, NY, USA, 1996. ACM.

[25] Antonio Si, Hong Va Leong, and Rynson W. H. Lau. Check: a document plagiarism detection system. In *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 70–77, New York, NY, USA, 1997. ACM.

[26] G. M. LaBeff E. E Vandehey, M. A. Diekhoff. College cheating: A twenty-year follow-up and the addition of an honor code. *Journal of College Student Development*, pages 468–480, 2007.

[27] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer Berlin / Heidelberg, 1999.

[28] Sven Meyer zu Eissen, Benno Stein, and Marion Kulig. Plagiarism detection without reference collections. In Reinhold Decker and Hans-Joachim Lenz, editors, *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 359–366. Springer Berlin / Heidelberg, 2006.