Lente

Looking for Equal or Near Text Efficiently

Grado en Ingeniería Informática

Departamento de Sistemas Informáticos y Computación

By:

Miguel Collado Segura

David Serrano Arce

Directed by:

Rafael Caballero Roldán

Authorization of diffusion and use

The signatories below, enrolled in the Grado en Ingeniería Informática of the Faculty of Computer Sciences, hereby authorize the Universidad complutense de Madrid (UCM) the right to use and diffuse this work for non-commercial, academic purposes, provided that said uses expressely quote the work's authors.

The UCM Library is also authorized by this document to deposit a copy of this work in the Archivo Institucional de E-Prints Complutense, with the purpose of increasing the diffusion, use and impact of the work in the web, and also to guarantee its access and preservation.

Date: June 19th, 2014

Miguel Collado Segura

David Serrano Arce

Thanks to Rafael for guiding us through this project and bearing with us during the process. To Victor for his help when we had almost given up on a part of the software.

Index

Abstract

Lente is a software capable of searching and detecting (by using the Internet) possible document plagiarisms. This program is able to analyze multiple types of plain text or PDF documents. It searches through the Internet, using a concurrent approach, for sentences and similar content to that of the document and allows the user to consult the results in real-time, before finishing. The results are web pages that can be opened from the very program, giving information about the exact content that is suspected of copy.

It is also able to work with multiple documents at the same time and show the individual progress of each one. There are different configurable search profiles, in order to optimize the time and resources used, based on simple parameters. This software manages a maximum number of simultaneous connections, and automatizes the use of proxies to boost the searching and avoid the limitations of use of the search engines used: Google and Yahoo.

Keywords:

Documents, text analysis, plagiarism, concurrent, web search, internet, real time results, proxies, Google, Yahoo.

1. Introduction

There are many free text-based search engines, like Google or Yahoo, that are the main way of interacting with the internet for a lot of people. These engines are free for a reason, they make a profit from data obtained during the search process. However, if one tries to find similar, more advanced services, such as looking for full documents instead of just a line (specially if we are looking for features like finding related, similar, or plagiarized texts), the set of free tools available becomes much smaller. Furthermore, the software available online is neither well-known nor effective in most cases. This is so because most of them rely on methods like using popular search engines to find some phrases from the document, which carries a lot of limitations, such as the terms of use of said engines, including their policies against *bots*. Another take on this issue would be to create a search engine devoted to this task, but the investment required to be able to compete with the better known ones makes it an almost impossible work considering the tool would be free.

With the motivation of having found only a scant amount of software that was both free and effective for this task, we decided to create a tool with all the features mentioned. Searching for document similarities on the internet has multiple uses: finding actual plagiarisms or quotations, texts with related topics or keywords, or finding the original source of a certain text from the internet. The limitation for all this is that the scope of the software will be the part of the internet that is indexed in the search engine chosen for each search.

State of the art:

The following are some of the free tools available on the internet in the scope mentioned, as of October, 2013:

· Grammarly (*http://www.grammarly.com*, in English): Checks for spelling mistakes and looks for possible plagiarisms. Allows a free revision of a document, but it only shows a very description of the results obtained without paying, displaying only the type of possible spelling mistakes and if

the text has been detected as a plagiarism of another source, but without giving any proof of it. The input can be provided by copying the text into a panel or uploading a file (doc or docx, only, does not support pdf files).

· The plagiarism Checker (*http://www.dustball.com/cs/plagiarism.checker*, in English): Checks phrases from the text provided on Google, showing them and wether possible proof of plagiarism has been found for each of them, showing the link of the relevant documents. It has a *Piremium*, paid version, which is advertised to be up to 3 times more accurate than the free one. A quick search using the free version and a text from the Wikipedia as imput (the day featured article) to test the software revealed that it could not identify the plagiarism, saying that the text was original. The document types allowed are .doc, and there is also the possibility to copy text directly into a panel.

· Plagiarism Checker (*http://smallseotools.com/plagiarism-checker*, in English): Analizes phrases from a given text, and provides links to the Google search used in each case to check wether each one is a plagiarism. It is completely free, without a paid version of any sort. Doing the same test as in the last tool, the software analyzed 28 phrases, out of which only two turned out as a *false negative*, stating that they were original. This tool does not support uploading files, using a “copy to text box” method.

· Plagiarism detect (*http://plagiarism-detect.com*, in English): Free to use. Allows searching by categories (essay, article, web page or other) Permite búsquedas por categorías (ensayos, artículos, páginas web u otros). Once the text is analyzed, the page shows the sources for the possibly plagiarized lines (as links). This detector looks for different sources for each phrase, in order to reduce the chance of outputting a *false positive* (stating that there is an unauthorized copy when there is not) to the user. However, doing the Wikipedia article test, the output was of only a 18% of copied text, saying that most of it was original. The allowed file types are .doc, .docx, .odt and .txt.

· Plagium (*http://www.plagium.com*, Multilingual): Searches using this program can be done in various languages separatedly, or even at the same time. It also provides the user with options like similarity threshold, search depth (quick or deep), or the scope (web pages, social notworks or news articles). The application requires a register to use, allowing each user to save the

results of their queries. The drawback to this software is that it uses a credit system. Thus, each search spends some credits, and in order to continue making use of this page one is required to buy more credits. Furthermore, these credits expire within a year of their purchase. The Wikipedia test returned a list of very complete and self-explaining results, with a 100% correct analysis, providing multiple web pages and quoting a part of the matching test for each result. The inputs allowed are raw text, webpage URLs and .txt and .pdf files.

· Detector de plagio (*http://detectordeplagio.com*, in Spanish): Free. It only supports plain text pasted into a textfield in the webpage, not allowing any kind of upload. No configuration options are provided, giving no information about the type of search done. The Wikipedia test using an article in english returned an “original text” result, stating (wrongly) that the text was original. A second test was done with an article from the spanish version of Wikipedia, but the results were the same, returning no found plagiarisms.

After taking a look into the available options in the field, we found two types of tools, according to the quality of their service: the free ones, often with results that could not be trusted, and the paid ones, which overall had a result quality relative to their price. Some of the most relevant issues found were the absence of multilingual support for the query (the search only working correctly for a certain language), the lack of the ability to upload text files from the computer or only allowing a few file types and the general bad quality of the results given (poor detection).

Objectives

To solve these problems, we decided on the following objectives:

- Allow any text-based file: To do this, the best option given our limitations is to accept pdf documents, since any text file can be converted to this format easily. Also, the software will be able to read txt files, since they are another commonly used format.
- Multilingual search support: Make a program that is able to search for plagiarisms in any language. Syntactic or lexical analysis will be avoided for this since the complexity of the code would be disproportionate.
- Multi-engine: Support different search engines, in order to get a wider variety of results.

In the next section, we will detail the most crucial parts of the project development, including the work model used, the requirements set, an explanation of the vital classes of the software and the main hardships encountered in the process.

Afterwards, a brief description of the finished software, the installation process, the program requirements and a brief user manual is presented. Also, some examples of use are described in this section.

Finally, the last two sections show the results, as well as the conclusions drawn from the whole project. In these parts, topics such as the required support for the software and its limitations, or the possible future upgrades to improve the software shall be explained.

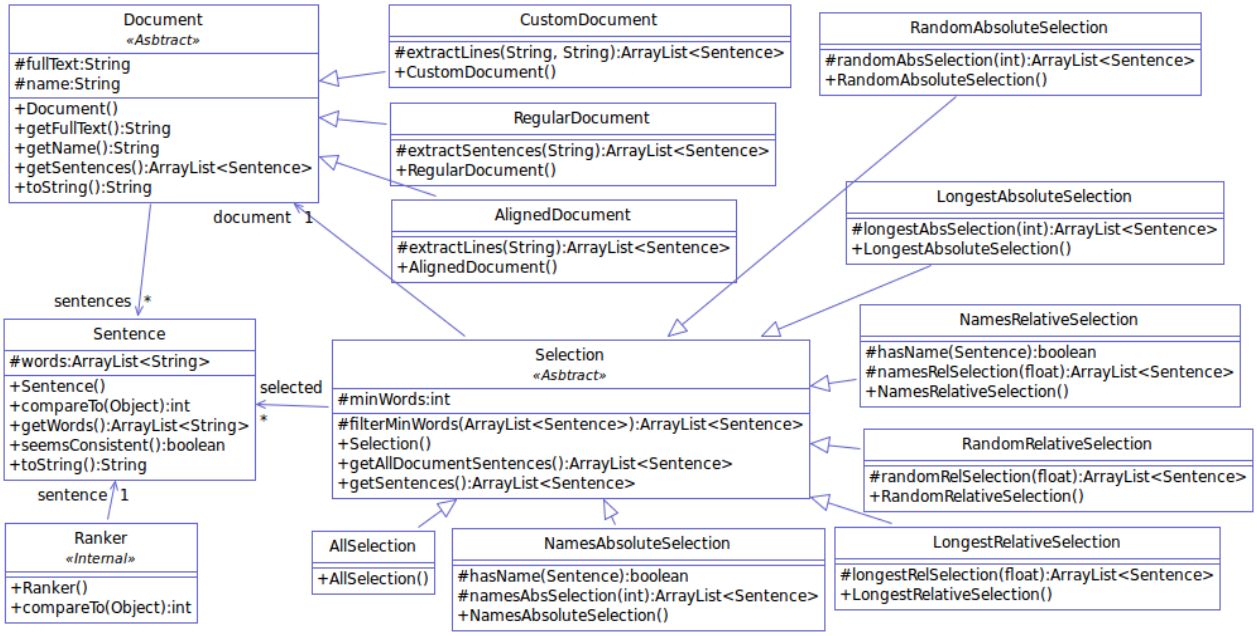
2. Project development

First, the following **requirements** for the software were set, taking into account the objectives listed before:

- Accept both plain text and PDF files as input.
- Look for web pages containing a certain text, obtaining the page link.
- Browse said pages in order to achieve a better accuracy (search engines may list web sites that are not relevant to the current query).
- Automatize queries and make them transparent to the user.
- Classify the shown output by its level of similarity to the input given.
- Allow simultaneous queries.
- Let the user check for the current results of any query, even if it has not finished yet.

Program structure

The following **UML diagrams** show how some of the key classes from the software will operate:



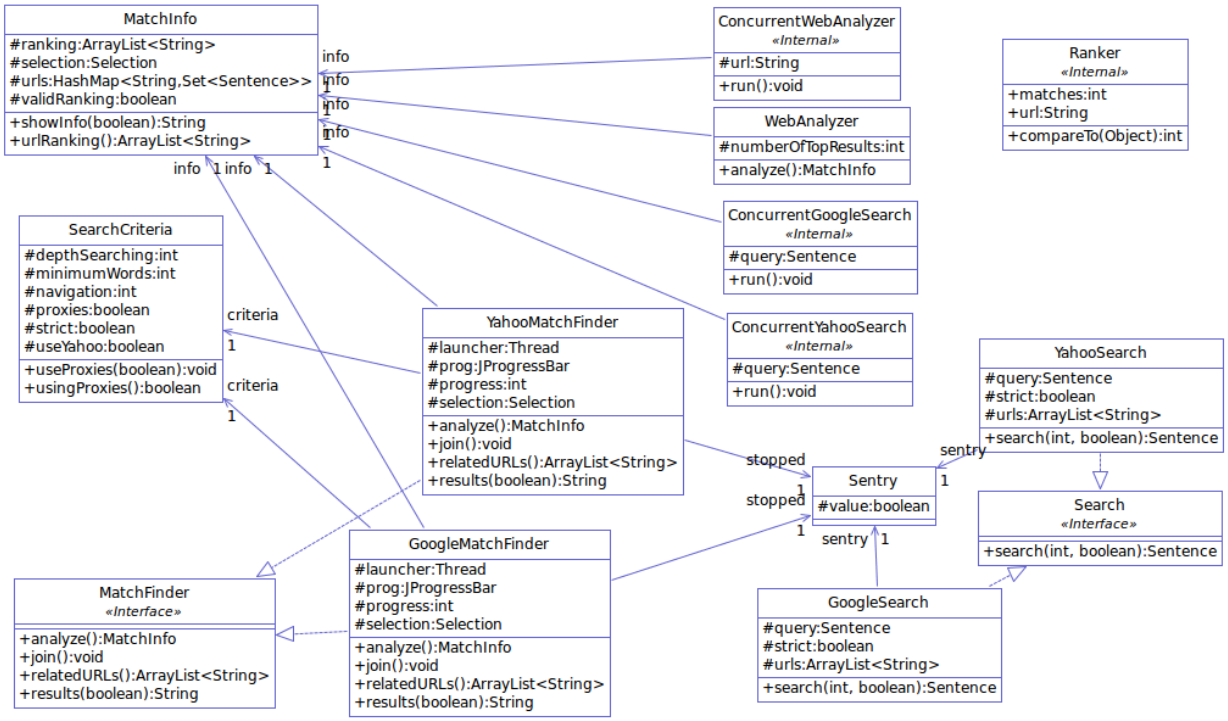
The Document and Selection classes serve the function of saving different kinds of documents, separating their sentences according to different criterias (the type of document), and then selecting the best ones to use in queries according to other set criteria (the type of Selection used).

Here, we will go into a more detailed description of each component:

- *Sentences*: a list of words. It is the basic element in the extraction and selection of text from a file. The text is split into them and then some are selected to create queries.
- *Document*: a container of sentences, linked to a file name. It will be used by the Selection to filter the best sentences for the queries. It also stores the full text of the file. Any implementation of how sentences are chosen must be done by extending this class.
- *RegularDocument*: extends Document in order to get sentences separated by dots.
- *AlignedDocument*: extends Document in order to get sentences separated by the newline character.
- *CustomDocument*: extends Document in order to get sentences separated by the given separator.
- *Selection*: a container for sentences selected based on any of the named criterias, and the Document from which said sentences have been *filtered*. It will be used by the search engine to get the sentences that will become queries. The kind of filter is up to the classes that extend this one. Additionally, this class filters those sentences that do not have a minimum number of words (which is configurable using the user interface).
- *AllSelection*: extends Selection to allow all sentences, making all of them into valid queries.
- *LongestRelativeSelection*: extends Selection to allow a percentage of the longest sentences, and discard the rest.
- *LongestAbosluteSelection*: extends Selection to allow a given number of the longest sentences, and discard the others.
- *RandomRelativeSelection*: extends Selection to allow a percentage of random sentences, and discard the others.
- *RandomAbsoluteSelection*: extends Selection to allow a given number of random sentences, and discard the rest.
- *NamesRelativeSelection*: extends Selection to allow a percentage of sentences which contain proper nouns (filled with more *normal* sentences if needed), and discard the others.
- *NamesAbsoluteSelection*: extends Selection, allowing a given number of sentences which contain proper nouns (filled with more *normal* sentences if needed), and discard the others.

- Ranker: a comparator used to sort sentences based on their length (number of words).

Search engine diagram:



The MatchInfo class gathers information from the query results in the shape of phrase-link pairs. Each MatchFinder is capable of filling a MatchInfo, according to the criteria given by the SearchCriteria class, using multiple threads to improve the efficiency of the process (by using the Concurrent...Search classes). Following the same multi-thread strategy, the WebAnalizer browses the best results from a MatchInfo. To achieve this, an implementation of a MatchFinder configured with a SearchCriteria (and adapted to a certain search engine) is launched. This MatchFinder returns a MatchInfo which can be accessed at any time, even while the query is still in progress.

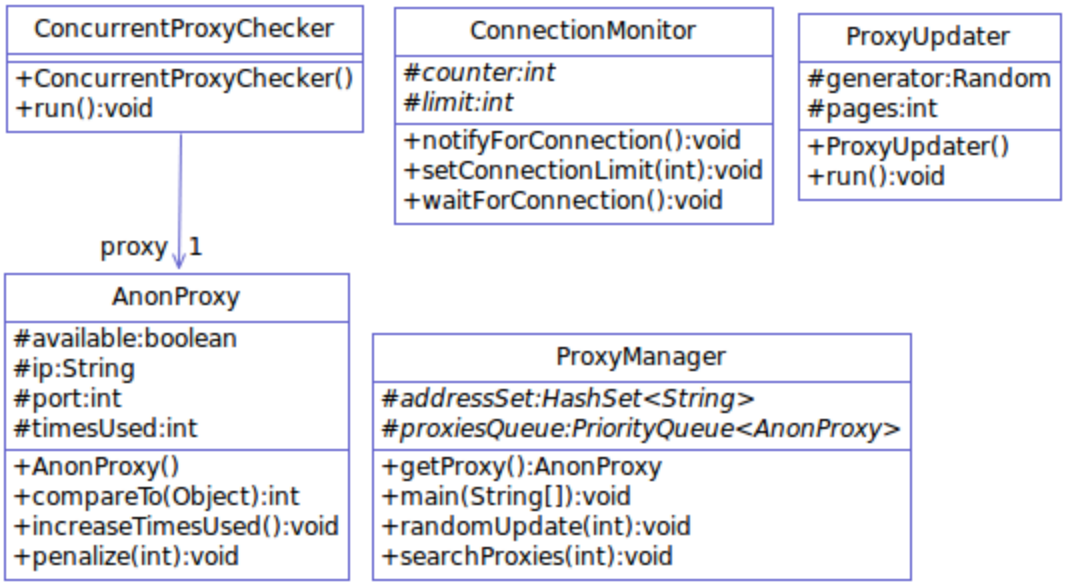
In more detail:

- MatchInfo: The core of the search engine. It stores pairs of sentences with URLs, also called *matches* This class is also capable to sort the *matches* to rank the URLs based on the number of *matches* of that URL with different sentences.
- SearchCriteria: contains properties of the search engine (which can be different each time), such as the minimum number of words per sentence, the number of results to

fetch per query or the number of best results to navigate. It also contains information about whether the use of proxies and the *strict* mode are enabled.

- MatchFinder: an interface whose implementations must realize a full search from a Selection and, based on the SearchCriteria given, fill a MatchInfo that contains the results of all the queries. Once finished, it will provide a list of URLs sorted by number of *matches*. Currently, there are two implementations (using different search engines): YahooMatchFinder and GoogleMatchFinder. Both share the same interface and the way they search with multiple threads: ConcurrentGoogleSearch and ConcurrentYahooSearch can resolve a query concurrently, so it is possible to run all of them at once and just wait for them to finish, which greatly improves the software efficiency.
- Search: an interface whose implementations must be able to resolve a query. both engines have implementations of it. Used by the Concurrent...Search classes previously mentioned.
- WebAnalyzer: navigates the best *matches* (a given number) concurrently, from a given MatchInfo, to improve the quality of the results: navigating the suspicious URL will find out the exact number of *matches*.
- ConcurrentWebAnalyzer: navigates a given URL concurrently and updates the given MatchInfo.
- Sentry: Keeps count of the number of threads that have been started. It is used for both the progress bar of each search and for the removal of a query threads if it is removed before finishing.
- Ranker: Comparator class to sort the results obtained.

Connection related classes diagram:



The connection management follows the diagram above. Since the number of queries each search engine allows is a limited resource, this program automatically makes use of proxies to avoid this limitation. The AnonProxy class saves the required information of one proxy and the number of times it has been used. The ProxyManager stores a set of proxies, and can be prompted in order to get the most promising one. It can also be programmed to automatically search for new proxies to use. The ConnectionMonitor is in charge of managing the maximum number of global connections available to the program (which can be altered) in order to avoid consuming an excessive amount of bandwidth.

Again, in a more detailed view:

- *AnonProxy*: contains a proxy (IP and port), its availability, and times it has been used.
- *ConcurrentProxyChecher*: checks if the given proxy is available.
- *ProxyManager*: contains a set of proxies, sorted by the number of times they habe been used (the less, the better), and provides the currently best proxy to be used. It updates itself by searching for new proxies. This class can provide proxies and penalize them concurrently.
- *ProxyUpdater*: searches for new proxies from the internet.
- *ConnectionMonitor*: monitor used to control the maximum number of global concurrent connections. All classes that needs to connect to internet, should use this monitor.

Work plan

In order to develop the software, we decided on an iterative and incremental model, which was inspired by the well-known SCRUM model:

The first stage of *product backlog* was done in October. From that moment, the plan was to do a series of mini sprints (by shortening them to seven day periods), with weekly meetings with our director (instead of everyday) in order to evaluate the current prototype and decide on the next steps to take, as well as possible ways to tackle any issues that had arisen. By March-April, we turned the project focus to fixing bugs and improving the interface, with weekly tests the program and the release of new betas.
During the whole process we made a point to maintain a regular communication in order to coordinate better and thus improve our teamwork.

Here is a brief description of the work schedule, as it went throughout the project:
[03/10/2013] First meeting, initial objectives and work strategy set.
[4/10/2013 - 17/10/2013] State of the art investigation on the free anti-plagiarism tools available.
[18/10/2013 - 28/02/2014] Incremental development of prototypes, including meetings every week in order to ensure that the program develops as expected.
[28/02/2014 - 25/04/2014] Final iterations of the prototype. Uploading of said prototypes to SourceForge in order to allow a wider sample of testers for the application.
[25/04/2014 - 23/05/2014] Final details and bugs polished.
[06/06/2014] Project end.

Technologies used

The following technologies were used in order to complete this project:

- Java 7:
  We chose Java as the programming language for this application because it allowed us

to port it easily to any operative system (since most of them can run .jar files from a console), and also because of the wide array of libraries available for us to use (we will mention some below). These two points, and the fact that the bottleneck in speed is caused in this case by the connection times (specially when using proxies, which were mandatory for us), made us take the decision of using this language instead of other finer grain solutions, such as the C programming language.

- Java Swing (to build the graphic interface):
  Since we were using Java, Swing was the most versatile library we could find for creating a GUI for the project. We took advantage of this library to create the whole graphic interface.

- JSoup (HTML parsing library):
  We needed an easy-to use solution for connecting to websites and getting their content. This provided us with the required number of samples to compare the text given to our software as input and determine whether it could be a plagiarism. This library offered exactly this text-fetching functionality and has proven to be both effective and user-friendly to us.

- PDFBox (a library used for decoding PDF files):
  In order to meet our objective of analyzing PDF files, we needed to find a library that allowed us to process the text from this kind of files without having to go through the problems of extracting it. Even though it proved to be complicated to learn to use, and failed to extract text in some cases, they were too few to deny the raw power of this library. While it does not support optical character recognition for pdf documents scanned as images, we would still use this tool over any other we tried while developing our software.

- FontBox (auxiliar library used by PDFBox to improve PDF compatibility):
  FontBox is a standalone library that improves PDFBox's font compatibility greatly, which allowed us to extract text from a wider number of samples, improving the software overall results.

Related courses

We wanted to briefly name the **courses** without which we would not have been able to do this software:

- "**Fundamentos de Programación**" introduced us to the main programming models and structures.
- "**Tecnología de la Programación**" increased our knowledge in Java and the Objective Programming paradigm, which we have based our work in.
- "**Estructura de Datos y Algoritmos**" showed us the most common data structures and some of the most important programming algorithms, without which we would not have been able to create a functional software.
- "**Programación Concurrente**" allowed us to discover how to create multi-threaded programs and deal with the problems that are inherent to the non-determinism they introduce. We found this course especially helpful because of an assignment which required us to create a simple spider software which grabbed links from websites. This inspired us when developing the web oriented part of our code.
- "**Ingeniería del Software**" taught us effective methods of working in order to develop functional software within a time period, choosing the best strategies and work flows in order to achieve the set goals.
- "**Inteligencia Artificial**" gave us an insight into natural language processing and its complexities, allowing us to decide how we would treat the input text for the software in order to get a sufficient set of phrases to look for possible plagiarisms.

Encountered Issues & Hardships

While in the process of creating this software, we encountered several walls we had to climb in order to succeed. Here are some of them:

Dividing the software functionality into two parts, analyzing text and efficiently searching for matches, the one which made us face the highest amount of issues throughout its development was, doubtlessly, the second. This was mainly due to the first one being less dependant on external resources, as will be explained below.

While we were in the text analyzing part, the main problems we encountered were caused by trying to find the answer to the following question: How could we read any text file? What we came up with was to make our software able to read PDF files, since any other file can be exported to it and every OS can read it easily*. Having decided this, we needed a way to extract the content from this kinds of files, which was solved by using the PDFBox library (http://pdfbox.apache.org), which is both free and functional. At this point, though, the results were still far from perfect, the problem being that many PDF files contain useless characters, use strange codifications, or are infact an image, making it impossible for us to extract the text from it with the tools available. To tackle this final issue, the best solution we found was defining first the criteria which would separate phrases in the text (by using the dot or end of line characters, or used defined separators) and then removing unnecessary characters, such as extra spacings or line breaks. This method proved satisfactory in most cases. It must be mentioned though, that in some cases the PDFBox library was unable to extract the text from certain PDF's, making us unable to analyze the text from them (this only happened on a very small percentage out of all the cases we tested).

Having finished this part of the functionality, we moved on to the next one, in which we encountered multiple issues, some of which we had not even thought we would encounter:

First, in order to be able to perform searches in the internet, we turned to the Google Search API (in JSON, https://developers.google.com/web-search), which we found out had been deprecated since November 1, 2010. The alternative provided by Google, CustomSearch (https://developers.google.com/custom-search) only provides 100 search queries per day freely, having to pay for larger numbers. We decided to test this limitation before actually giving up on using the API, but the result we obtained was the following exception once we did some searches:

Caused by: com.googleapis.ajax.services.GoogleSearchException: **Suspected Terms of Service Abuse**. Please see http://code.google.com/apis/errors

Looking for the actual price (to see if it could somehow be within our limitations) of the extra queries, we were faced with the sum of $5 for every 1000 queries, which was unthinkable in our

conditions (specially since our program requires a considerable amount of queries for each document searched).

This problems forced us to take a different route in the development: searching the web directly, as we would by using an internet browser (using the jsoup library to aid us in the process). For this, we needed to be able to translate any phrase into search link. Since URLs are limited to ASCII characters, we had to investigate how said URLs were created when searching (by using a trial and error method). We solved this considerably fast, discovering that first the phrase had to be translated to UTF-8, where each character follows the %XX or the %YYY formats, and spaces are replaced by '+' symbols. Having solved this, we still needed a way to test it.

We did not come upon the way to extract the links from Google result pages by the last stages of the development, and so we decided to look for other alternatives, such as Yahoo (in which we encountered the same problem as with Google, having to find a way to extract the links from their result pages manually) or Faroo (which advertised their free web API, but offered results that were of almost no use to our software). By this point, we managed to discover a way of getting Yahoo results, being able to do a big amount of searches before encountering an exception like the one below:

org.jsoup.HttpStatusException: HTTP error fetching URL. **Status=999**, URL=http://search.yahoo.com/search?p=...

This became another issue in our growing list, but this engine could at least be used for testing purposes, letting us refine the software functionality (while having to reset our computer IP address from time to time to continue working).

By April 2014, our software was already capable of searching plagiarisms using Yahoo search engine (and avoiding the exception mentioned before) and browsing the output to refine the results given to the user. At this time, while we were still trying to unravel Google way of showing its query results, we suddenly were faced with an unexpected problem: Yahoo had changed its interface and thus we had to rewrite most of the code we used to parse it.

When we finally succeeded in getting the URLs from Google result pages (which proved especially difficult because it used redirects instead of direct links), we had already come up with a system to avoid both limits for the number of searches allowed. To do this, we came up with

the idea of using proxies, since the limitation was actually based on the computer's IP address. The system we created obtained said proxies in a dynamic way, getting a self-updating list of available ones and making connections through them. To speed up the connection as much as possible (since using proxies proved to be somewhere between 10 and 100 times slower than direct connections), the system penalized the slower or "used up" proxies (the ones which would not allow more connections to Google or Yahoo), so connections would always be made using the most promising ones. Thanks to this, we could finally have a virtually unlimited amount of queries without fear of having our search interrupted before ending.

3: Application

Installation

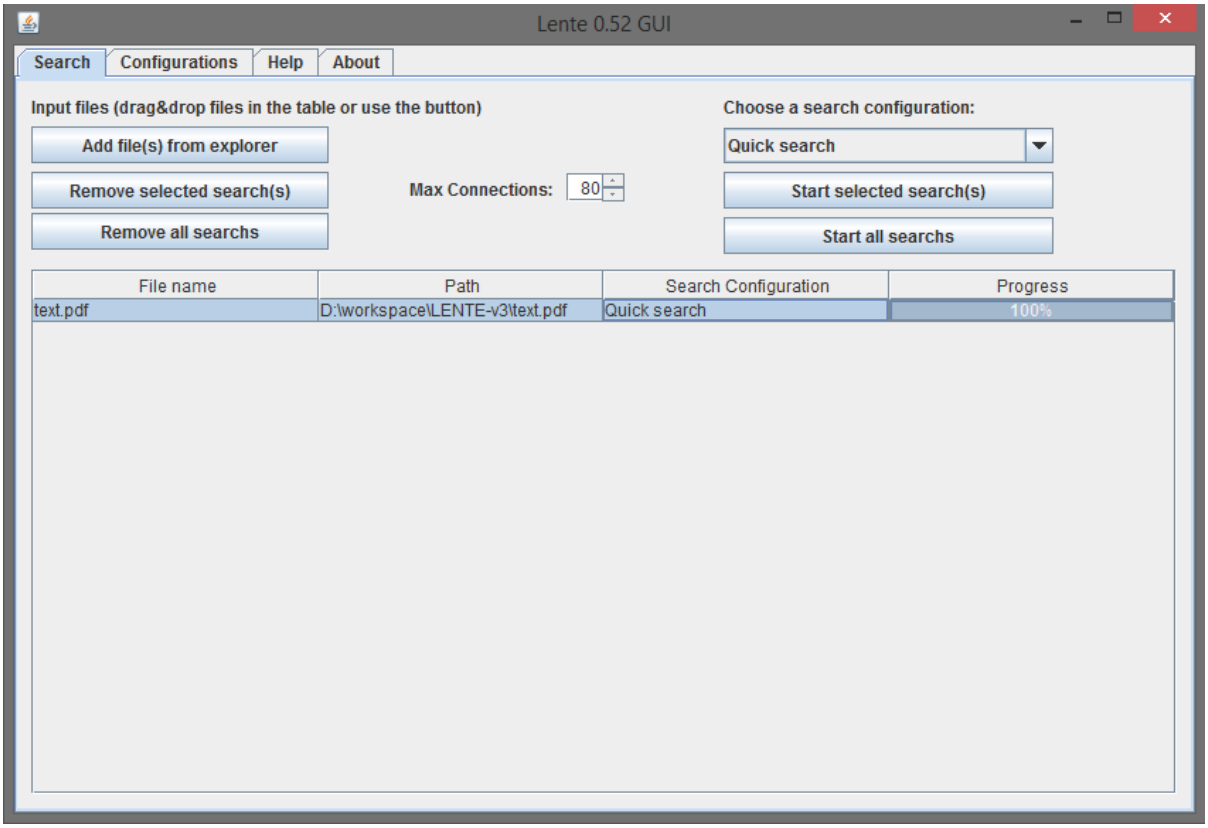
LENTE is a portable application, and so it does not require any type of installation. To run it, simply double click the lente.jar file (on Windows), or open a console and use the “java -jar lente.jar” command on Linux based systems.

Requirements

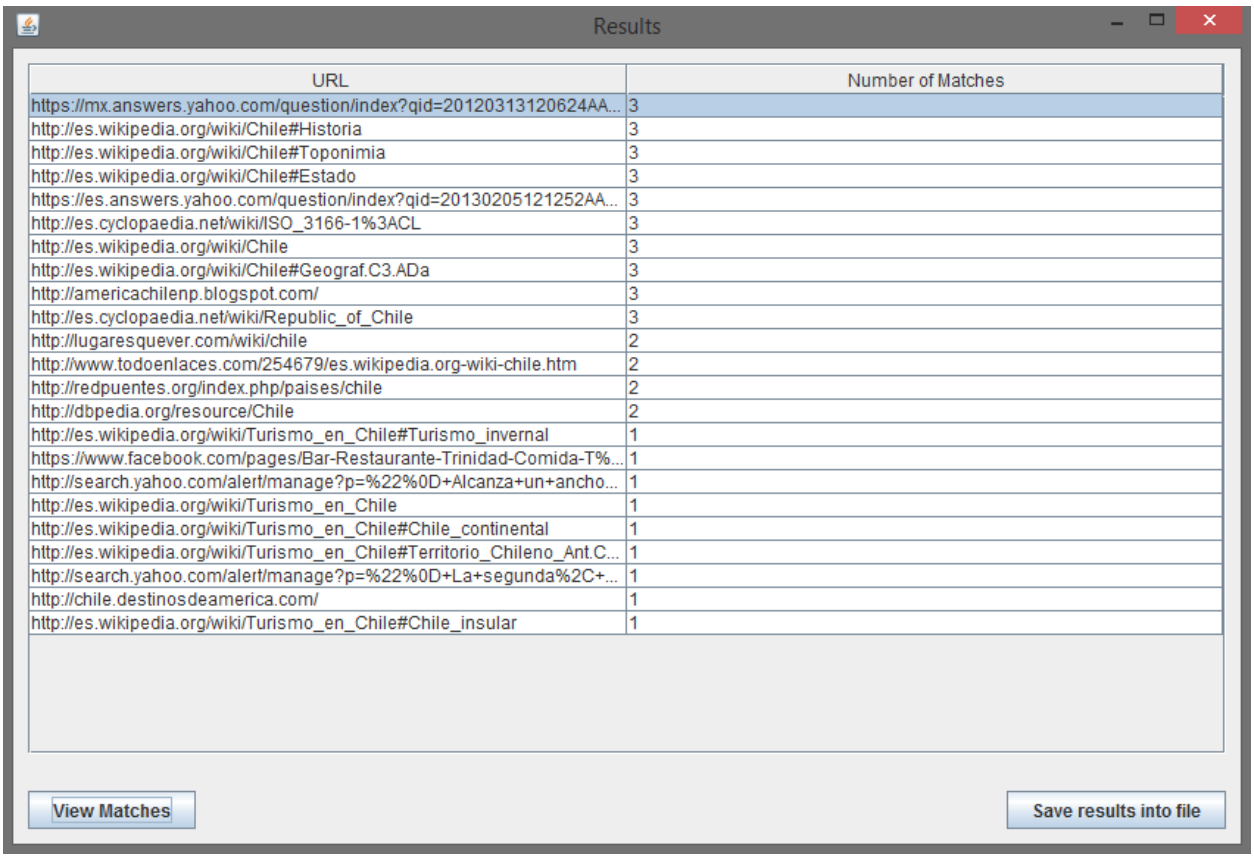
LENTE requires a Java Virtual Machine to run. Java can be downloaded for free at their official website (https://www.java.com/es/download/). Also, the software needs an internet connection in order to work.

User Manual

LENTE has a simple interface consisting of 2 main tabs (plus a third one for quick help and a last one with contact information).
The first tab, called “Search” is where the user can make their desired queries. The process will be explained below:



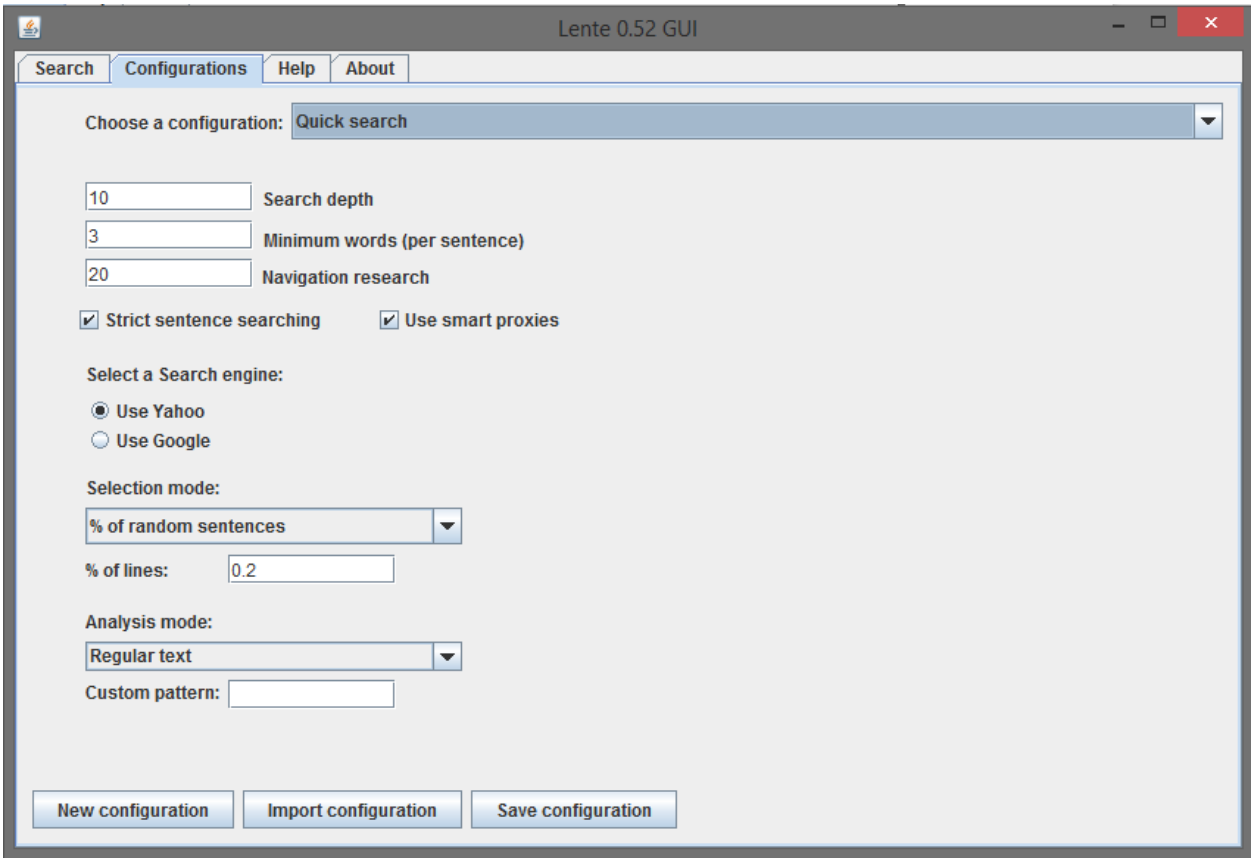
- The user can add files to be searched by using the “Add file(s) from explorer” button or by simply dragging & dropping the desired files into the table occupying the bottom part of the window.
- To start searching for plagiarisms, the user can either press the “Start all searches” button or select the desired files from the table and use the “Start selected search(s)” button instead.
- Once a query has started, its progress bar will start advancing. At any time from that point, the user may now double click it on the table to open the results window. Note that said window will not refresh by itself, and so it must be closed and reopened in order to see the changes as the search advances. An example of a “results” window can be seen below:



- In this window, a double click on a result will open its link in the computer default browser. Clicking the “View Matches” button will show the exact matching phrases that were found in a new window.
- Back in the main window, to remove an ongoing (or finished) query, the user can use the “Remove selected search(s)” button after having selected it on the table (multiple selecting is allowed) or the “Remove all searches” if none of the current queries wants to be kept.
- The “Max connections” selector allows the user to set the maximum number of connections to the internet the software will be able to do at a given time. A bigger number in this setting will result in a better performance, but will also make the software consume a bigger amount of bandwidth.
- The search configuration dropdown selector on the top right lets the user choose different configurations, as set in the “Configurations” tab (which will be explained later on). Please note that this configuration will be applied to new searches added, and not the ones that

are already on the table, so it should be changed before adding a new query when desired.

The second tab, “Configurations” lets the user change a variety of settings in order to tune the program to their needs:



- From the dropdown selector on the top, the user can select the configuration to be edited. To create a new one, the button “New configuration” on the bottom left corner can be used (which will prompt the user for a name). Then, the new configuration will be available to be selected on the dropdown list.
- The “Search depth” field refers to how many results (in the search engine) are searched for each selected sentence from the text of the document. The “Minimum words” one filters sentences so not one with less than the specified number of words will be considered for searching. Finally, the “Navigation research” field refers to how many of the results with the most matches will be browsed to obtain a better analysis of possible plagiarisms.

- The “Strict sentence searching” checkbox controls whether the program will look for exact full phrase coincidences or a more lax matching (which may be specially useful for looking for similar texts instead of plagiarisms). Next to it, The “Use smart proxies” one toggles the use of proxies for the search engine queries. We strongly recommend leaving this setting on in all configurations, since deactivating it may block your computer’s IP address temporarily from using the chosen search engine and the search is not guaranteed to finish.
- The “Selection mode” dropdown menu lets the user select which sentences from the text will be searched for possible plagiarisms. The number or percentage of sentences to look for must be specified in the empty field right below it, and if the desired method is “All sentences”, the field must be left blank. Please note that percentages must be written as a number between 0 and 1 (for example, 0.3 would be the same as 30%).
- The last dropdown menu, “Analysis mode”, lets the user specify how sentences should be interpreted from the input text. “Regular text” will separate sentences by each dot (“.”) in the text, whereas the “Aligned text” will separate them by each line. A custom separation pattern may be set by using the menu third option, and filling the field below it with the desired separator (regular expressions are accepted). This field must be left blank otherwise. To learn more about regular expressions, we recommend visiting this website: http://www.regular-expressions.info/java.html

Examples of use

Here, we will explain the best configurations for some hypothetical scenarios:

Checking for possible plagiarisms in a long History essay, in pdf:

In this case, the fastest solution is to do a Quick Search (selecting it from the dropdown menu in the Configurations tab).

A search depth of 10 to 12 results per sentence should provide enough documents to compare to. In this case, we will set this parameter to 11.

Since the document is an essay, sentences with less than 3 words can be omitted, as the more relevant ones will probably be longer. Thus, we will leave the Minimum words parameter as-is.

Both checkboxes (strict sentence searching and smart proxies) shall be selected. This is done in this way because of the document being long, which implies that the chances of literal copies increase. If this method fails, a second search could be done unchecking the strict sentence searching option.

The search engine is not really relevant, but Google often finds better results, at the cost of taking a significant amount of time longer to finish compared to Yahoo.

Finally, we will set the selection mode to “% of longest sentences”, and set its parameter to 0.3 to select the 30% of the lines from the text, ordered by its length. Also, the analysis mode will be left as-is, in the “regular text” selection.

Looking for related articles to a short publication:

In this case, we will use the Deep search preset, since the text is short.

The Navigation research parameter will be set to 15, since we don’t need exact matches, but only similar ones and thus, a smaller body of documents to compare to can be used.

The strict sentence parameter will be left unchecked, since we want to find related articles, which do not have to necessarily have fully matching sentences.

The selection mode will be left to All sentences (since the article itself is short), and the Analysis mode shall be left as well as regular text.

Checking a C++ code suspected of not being original:

The most important point in this case is to set the analysis mode to ”Custom sentence separation”, since in this case we want lines to be separated by the “;” character. Thus, in the Custom pattern space, we will put a ; symbol.

The other parameters will depend on the length of the code, but we recommend using the Deep search preset with the modification described above.

4: Results

Main outcome

The resulting program is an application available as a .jar file. Java 7 or higher is required for its functioning, and no other dependencies are needed. The current version (as of the writing of the document) is 0.61, adding up to a total of more than 220 downloads (from over 25 countries around the world) as freeware on Sourceforge.[1] Since its only requirement is Java, it can be run in Windows, Mac OS and Linux.

Limitations

Most of the limitations explained here come from the way we solved the issues noted before, or from the impossibility to solve them completely. They are as follows:

First, there's a number of PDFs that we have no possibility of reading, since using optical character recognition was out of our limitations (for the ones composed of images), and also because PDFBox failed, as we mentioned earlier, to extract the text from a small number of documents.

Also, the software has a strong dependence on the proxies available and having a reliable connection (since an unstable connection affects negatively to our proxy rating system). Even though searches can be done without the use of proxies, its number is very limited and can lead up to the search engine blocking the user IP address from its services for some time, which is something we could not allow.

Furthermore, as we also explained before, the program relies on the current web interface and internal code of the supported search engines, which means that any change done in them could stop it from working until we adapted its code. This is perhaps the strongest limitation in the software, since we are forced to stay on the lookout for changes on said interfaces.

[1] The project's page is https://sourceforge.net/projects/lente/

Finally, the search process cannot be paused, only stopped, and even this takes a considerable amount of time since the program has to wait for the proxies to answer its queries.

Support

This software support is centered around its dependencies. If the web interface of any of the search engines used were to change, we would be forced to rewrite or modify the code in charge of parsing the result links. These changes can happen at any moment, and there is no actual guarantee that they will not occur often (as we mentioned earlier, we already encountered this issue once while developing the program). Also, there is a possibility that the systems used to detect users evolve, forcing us to create a new mechanism to enable our program to keep doing searches without a limit, adapting to the changes made every time.

The other strong part of the program's support would be focused on the sources we use to get new proxies, since any change in their format would enable our software main functionalities until the code was patched, only allowing direct searches.

5: Conclusions and Future Work

During the development of this project, we strove to create a software that was, above everything else, useful. Along the process, we faced a number of problems and challenges, which forced us to find alternate ways of solving them in order to move forward. The process was sometimes frustrating, but we are now proud of its results.

As we have pointed out in some parts of this document, we released our program as *freeware* as soon as we felt it was ready to be used, since it would not only give us a bigger chance of finding possible bugs, but also because our objective was always for the software to be a practical tool available to everyone.

Given the tests we have done, we think we have succeeded in this point. By the moment of the writing of this document, we have over 220 downloads on SourceForge, and that number is further increased by the copies we have provided co-workers and fellow students. We have had a very nice amount of feedback, and given the results we collected, the program was actually useful for other students, helping them check their work and look for related material on the web. Although the amount of teachers using our program (that we know of) was lower, the feedback in this case was also positive, helping them in their work.

The most enriching part of this whole experience was to learn about a field that was at first unknown to us, and getting to understand why there are so little tools capable of fulfilling the goals we set for this project. The external resource dependencies require a constant vigilance, as any change made by the search engines we use could potentially render the program useless, as we mentioned in the Support section.

Also, the development plan we decided on was really helpful while constructing the software: the weekly meetings with our director helped us point the work in the right direction, and the short iterations helped us to center on tasks one at a time, ensuring each worked as it should before moving on to the next one. Also, the decision to publish the program helped us find issues that had escaped our filters, contributing to make a better software overall.

Finally, in order to improve the software beyond this point, we have set some upgrades that could be done in the future:

- Accept other types of documents: .doc and .docx files are widely used and being able to use them would help our software versatility. Another interesting input to accept would be web pages as a whole, scanning them and looking for possible plagiarisms in their text. This would be specially useful, for example, for a news company in order to ensure that their articles are not being stolen.
- Search for possible plagiarisms in a closed set of documents (instead of looking for matches on the internet). This would have direct uses in teaching, allowing teachers to check their student works by cross-comparing them.
- Semantic searching. This might be the most important point, because it would enable the software to do much more precise searches, instead of being limited to a search engine's "literal" or "similar" keyword searching. Analyzing the text and doing a natural language interpretation, however, is a complex matter and would require a considerable effort.
- Ignore valid quotations when looking for possible plagiarisms.

The sum of all the factors explained allow us to understand why having a professional plagiarism detector is a concept that is close to a utopy. Text analysis, language translation, syntax interpretation and having fast yet reliable queries are all functionalities that require an enormous effort to accomplish. If we add the fact of the software being free to this mix, we find ourselves in front of an enormous wall to overcome. Using a paid software model, on the other hand, would have made the querying process a lot simpler, allowing us to spend more time in other parts of the software. An example of this is the Turnitin tool, which is a remarkable software in its field.

This, however, was incompatible with our goals (and also introduced other difficulties, as finding an initial funding), so we decided to stay with our original idea. We believe that, with this work, we have proven that sometimes the simplest ideas turn out to be powerful enough to be of use. A relatively small, local piece of software that uses the strenght of well-known engines turned out to be able to accomplish the titanic task of discovering similar documents in some corner of the internet.

Spanish Section

Resumen

Lente es un software de detección y búsqueda vía internet de posibles plagios de documentos. Puede analizar diversos tipos de documentos de texto o PDF. Busca a través de internet, de forma concurrente, frases y contenidos similares al documento y permite consultar los resultados en tiempo real antes de finalizar. Los resultados son páginas web que pueden ser accedidas desde el programa, permitiendo consultar qué contenidos han sido supuestamente plagiado.

Puede trabajar con varios documentos simultáneamente y mostrar el progreso individual de cada uno. Ofrece diferentes perfiles de búsqueda configurables, para optimizar el tiempo y recursos utilizados, basados en parámetros sencillos. Gestiona un número máximo de conexiones y puede gestionar proxies de forma automática para acelerar la búsqueda y evitar las limitaciones de uso que dan los buscadores utilizados: Google y Yahoo.

Palabras clave:

Documentos, análisis de texto, plagio, concurrente, búsqueda web, internet, resultados en tiempo real, proxies, Google, Yahoo.

Introducción

Existen muchos buscadores de texto gratuitos, como Google o Yahoo/Bing, que constituyen el medio principal a través del cual muchos usuarios utilizan internet. Son gratuitos porque para estos buscadores es rentable haciendo negocio de los datos obtenidos de esas búsquedas. Sin embargo, si se busca otros servicios similares, como podría ser buscar archivos enteros de texto en internet, para buscar documentos relacionados, similares o directamente plagios, se encontrará con que no hay muchos. O mejor dicho, los pocos que hay no son muy conocidos

y/o eficaces, y la mayoría se basan en buscar frases o fragmentos de texto en los buscadores habituales, con los problemas que eso supone: depender de otros servicios y sus cambios, las limitaciones de sus términos y condiciones, y las limitaciones reales de sus sistemas de cortafuegos ante *bots*. La otra opción es crear tu propio buscador enfocado a esta tarea, pero para tener el poder de indexación capaz de competir con los buscadores clásicos, requiere una inversión en recursos que haría imposible crear una herramienta gratuita.

Con la motivación de haber encontrado una escasa cantidad de utilidades gratuitas y eficaces para esta tarea, creamos esta herramienta con esas cualidades. La búsqueda de similaridades en internet de archivos tiene múltiples utilidades: encontrar plagios o citas (plagios parciales hablando técnicamente), encontrar documentos relacionados por temática o palabras y encontrar el origen de algún documento que procede de internet. Aclaremos que el ámbito real será la parte de internet que está indexada en el buscador que se utilice para buscar.

Estado del arte

Éstas son algunas de las aplicaciones que hay en este campo (invastigación realizada en Octubre de 2013):

· Grammarly (*http://www.grammarly.com*, Inglés): corrige ortografía y busca posibles plagios. Permite revisar de manera gratuita un documento, pero sólo ofrece una descripción del resultado del procesamiento bastante pobre de manera gratuita, simplemente indicando el tipo de errores gramaticales posibles y si el texto es o no copiado de otra fuente, sin ofrecer pruebas o explicaciones al respecto. Permite copiar el texto en una ventana para analizarlo o subir archivos desde el ordenador (no acepta PDFs, abre DOC y DOCX).

· The plagiarism Checker (*http://www.dustball.com/cs/plagiarism.checker*, Inglés): busca frases del texto introducido en Google, mostrando las frases buscadas y si se han encontrado posibles pruebas de copia, mostrando enlaces de las coincidencias encontradas. Tiene una versión *Premium* de pago, que anuncia ser hasta 3 veces más precisa que la gratuita. Una búsqueda rápida con un trozo de un artículo de Wikipedia para probar la eficacia del buscador devolvió un resultado negativo (no se sospechaba copia alguna). Permite introducir texto plano en un cuadro o subir archivos DOC.

· Plagiarism Checker (*http://smallseotools.com/plagiarism-checker*, en Inglés): analiza frases del texto introducido para buscar posibles copias. Es completamente gratuito, y una vez finalizada la examinación, provee de enlaces a búsquedas en Google para las frases que determina como copiadas, permitiendo verificar si es el caso. Realizando la misma prueba que con el anterior buscador, los resultados son los siguientes: de 28 frases analizadas, sólo dos de ellas dan un *falso negativo*, diciendo que no son copiadas de forma errónea. Las demás frases son correctamente identificadas.

· Plagiarism detect (*http://plagiarism-detect.com*, en Inglés): Permite búsquedas por categorías (ensayos, artículos, páginas web u otros). Su uso es gratuito. Una vez analizado un texto, muestra enlaces a las fuentes de los posibles plagios encontrados. Busca una misma frase en varias fuentes, reduciendo la posibilidad de que una frase detectada como copia no lo sea. Dicho esto, haciendo la prueba con el artículo de Wikipedia, sólo encuentra un 16% (en lugar del 100% que es) de texto copiado. Permite subir archivos .doc, .docx, .odt y .txt.

· Plagium (*http://www.plagium.com*, Multilenguaje): permite hacer búsquedas en varios idiomas por separado, o incluso a la vez. También proporciona al usuario opciones de búsqueda como el umbral de similitud, la profundidad de la búsqueda (rápida o profunda) o dónde buscar (web, noticias o redes sociales). Requiere un registro para su uso, y con ello permite guardar búsquedas en la cuenta usada. Por otro lado, se asigna una cantidad de créditos a dicha cuenta, que se gastan con cada nueva búsqueda realizada y que además caducan tras un año. La única forma de obtener más es realizando pagos. Tras una búsqueda de otro texto copiado de Wikipedia, los resultados obtenidos son explicativos y muy completos, obteniendo un 100% de coincidencias en varias páginas web, de las cuales el motor provee el texto con las partes coincidentes resaltadas para facilitar la refutación.

· Detector de plagio (*http://detectordeplagio.com*, Castellano): gratuito. Únicamente acepta texto plano pegado en un cuadro dispuesto en la página para tal fin. El texto en inglés usado en los otros buscadores como prueba fue detectado como original, lo cual no aporta mucha fiabilidad a sus resultados. Una segunda prueba con un texto español (el mismo que en el buscador anterior) tampoco reportó resultados de plagio positivo. No ofrece ningún tipo de personalización ni configuración para su uso.

Visto el panorama general de las aplicaciones existentes, encontramos básicamente dos tipos según la calidad de los resultados: los servicios gratuitos de dudosa eficacia y los servicios de pago de calidad proporcional al precio. Algunos problemas generales son más importantes son la ausencia multilingüismo (que la búsqueda sólo funciona en algún idioma concreto y no lo demás), la falta de compatibilidad de archivos de texto (limitación a algunos formatos o simplemente un cuadro de texto en el que introducir el texto, lo cuál es incómodo para grandes volúmenes de texto) y malos resultados en general (poca capacidad de detección de copias).

Para solucionar esos problemas, marcamos una serie de objetivos:

- Multi-formato, que pueda leer de cualquier tipo de archivo con texto. La mejor forma es leer PDF ya que cualquier archivo se puede convertir en PDF.
- Multi-lenguaje, que pueda trabajar con cualquier idioma. Necesitará alejarse del análisis sintáctico o semántico (eso requeriría un aumento desproporcionado de la complejidad del programa).
- Multi-buscador, dar la opción de buscar a través de varios buscadores y tener más variedad de resultados.

Conclusiones y trabajo futuro

Durante el desarrollo de este proyecto, nos hemos esforzado en crear un software que fuera, por encima de todo, útil. Durante el proceso, hubo que afrontar numerosos problemas y desafíos, que nos forzaron a buscar formas alternativas de resolverlos para poder avanzar. A veces resultaba frustrante, pero estamos orgullosos de los resultados.

Como hemos indicado en algunas partes de esta memoria, hemos publicado nuestro programa como freeware tan pronto como sentimos que podía ser usar, ya que no sólo nos daría mayores probabilidades de encontrar bugs, sino porque nuestro objetivo fue siempre ayudar a la comunidad.

Dadas las pruebas que hemos realizado, creemos que hemos conseguido este objetivo. En el momento de la escritura de este documento, nuestro software cuenta con más de 200

descargas en SourceForge. Además, a este número se añaden las copias que hemos distribuido por otros canales a compañeros de clase y conocidos. Gracias a ello, hemos recibido una gran cantidad de feedback, y dadas las opiniones recogidas, podemos afirmar que nuestro programa ha resultado ser una herramienta útil para su trabajo, ayudando a estudiantes con sus trabajos, por ejemplo, asistiéndoles en la búsqueda de documentación relacionada con su estudio. Por otra parte, si bien el número de profesores que han utilizado la herramienta (y que tengamos constancia de ello) ha sido menor, en este caso también hemos recibido evaluaciones positivas acerca de su utilidad.

Lo más enriquecedor de toda esta experiencia ha sido aprender de un terreno desconocido en un primer momento, y acabar entendiendo por qué hay muy poco software capaz de hacer eficientemente la tarea que el nuestro hace. La dependencia de los recursos externos requiere completa vigilancia de los cambios que puedan hacer los buscadores, y para colmo, nadie te garantiza que podrás reaccionar ante el cambio y que todo siga funcionando.

Por otra parte, el plan de trabajo que elegimos resultó ser muy útil para ayudarnos durante el desarrollo del programa: las reuniones semanales con nuestro director nos ayudaron a mantener el rumbo del proceso en la dirección correcta, y las iteraciones cortas a centrarnos en una sola tarea a la vez, asegurando con ello que cada funcionalidad cumplía con lo esperado antes de pasar a la siguiente. Además, la decisión de publicar el resultado de nuestro trabajo como *freeware* resultó ser de una gran ayuda para nosotros, puesto que expandió enormemente nuestro banco de pruebas, permitiéndonos encontrar fallos en el software que se habían escapado a nuestros filtros, contribuyendo con ello a conseguir crear una mejor herramienta.

Para mejorar el software más allá de su estado actual, hemos propuesto algunas mejoras que podrían ser hechas en el futuro:

- Aceptar otro tipo de documentos: los archivos .doc y .docx son ampliamente usados y ser capaces de usarlos ayudaría mucho a dar versatilidad a este software. Otra forma interesante de poder introducir documentos sería, a través de su URL, aceptar páginas web, que podrían ser escaneadas y así poder extraer su texto para ser analizado. Podría ser útil para editoriales, por ejemplo, ya que podrían buscar artículos que les hayan podido plagiar.

- Búsqueda de posibles plagios en un conjunto cerrado de documentos (en lugar de buscar a través de internet). Tendría usos directos en la enseñanza, permitiendo a los profesores comprobar los trabajos de sus estudiantes comparándolos directamente
- Búsqueda semántica. Esta sería quizá la mejora más importante, porque permitiría a este software ser mucho más preciso en sus búsquedas, en lugar de verse limitado por un motor de búsqueda de palabras “literalmente iguales” o “similares”. Analizar el texto y hacer una interpretación del lenguaje es, sin embargo, una materia compleja que requeriría un esfuerzo considerable.

La suma de todos estos factores llevan a entender por qué tener un detector de plagios profesional es casi una utopía. El análisis del texto, la traducción de otros idiomas, la interpretación sintáctica, exigir rapidez a la búsqueda sin perder fiabilidad… requieren un esfuerzo enorme que de por sí, cuesta mucho dinero desarrollar y mantener un software local así (haciendo la gratuidad de éste imposible). Es más fácil (y evita problemas legales) tener un servicio online que haga las búsquedas, como hemos visto en el caso de Turnitin, y cobrar por él; para poder mantenerlo más fácilmente y regular su uso.

Pero con nuestro proyecto hemos demostrado que a veces lo más simple también funciona bien, y un sencillo software local que aprovecha la fuerza bruta de los buscadores más potentes que hay, a la vez que gestiona las limitaciones que nos imponen, es capaz de hacer una tarea tan titánica como descubrir si en algún rincón de internet hay algo que se parezca a cualquier parte de mis documentos.

Addendum: Each Student's Contribution

Miguel Collado Segura:

My contribution to this work consisted in various tasks, which will be listed and briefly explained below:

-Research on the field's state of the art: In order to determine what we wanted our software to do, we first needed to do an analysis of the current tools available, their strengths and their flaws, in order to make a program that would be of use to everyone, improving the options that they can use.

-Research in order to find possible libraries and tools to use by our program: One of the main reasons for using Java as our development language was the wide selection of libraries we could use. We needed to find tools that helped us in parts of the process that were already coded, such as the parsing of PDF files by using the PDFBox library, or the JSOUP parser which helped us get the actual links from Google and Yahoo's result pages.

-Translation, design and writing of a great part of this document: We wanted our program to be available to as many people as possible. We decided that, in order to achieve this, this document should be written in English, since the language is more spread world-wide than Spanish.

-Publishing of the work on SourceForge, advertising of the software and providing end-user support: As mentioned in the last point, we wanted this tool to be available to as many people as possible, and so we decided to publish it in SourceForge as freeware, allowing anyone to download, use and ask any doubts they had via this platform. This also required the setup of a mail address dedicated to answering support questions and issues to make it easier for any user to contact us. Also, to improve the software's visibility, I used social media to advertise it,

allowing more people to get to know of the existence of this program. This last point was specially effective, since an important number of fellow students from other fields of study used our software and gave us feedback about it, which helped us greatly.

-Implementation of the connection via proxies: We needed to be able to use search engines without getting the user's ip banned because of making too many queries in a short amount of time, so we resolved on using proxies as a channel to mask the user's IP address, allowing the program to run much more efficiently. This way, the program uses proxies from a list that is updated constantly, discarding those that take too long to answer our queries or that get banned from the search engine in use.

-Coding of the Google queries and its results parser: This task required us to rethink the model we used for the Yahoo engine, since Google's links were not displayed as clearly. This made us spend a lot more time than intended on it. Even though the parser is different to the one designed for Yahoo, the core template used to guide its development was the same.

-Implementation of the selection filters: We needed filters that would get the most relevant information from the text parsed in a simple way in different scenarios. To do this, we decided on implementing filters based on each sentence's length, and also on the use of proper names, allowing the user to choose the best fitting one for their needs. Every criteria includes a version based on a percentage of the total number of sentences (intended for longer documents), or one based on a given number of sentences to get.

-Design, coding and refining of the User Interface: Once we had a working software, we needed to create an interface that would be intuitive and easy to use. With this purpose in mind, I created one based in tabs, which most users are familiar with. I also used a design similar to torrent programs for the main tab, with a table showing the running searches and their progress, updated in real time (this last part was especially difficult since progress pabrs have to be rendered separately in order to be shown correctly). In order to solve any doubts that a new user might have, I also included a help tab with examples and explanations of the different parts of the interface, starting with an explanation for a "quick use", which only requires the user to drag & drop files to the search table and click the search button. There's also an "about" tab with

information about us, and where to contact us for support. The configurations tab was designed to be simple to understand, and to already have a set of defaults that will work for general uses.

-Testing and debugging of the main program: In order to ensure that the software worked as intended, I did a series of tests for every release we developed, and also coordinated the support section on a later time when we published the software, which gave us a wider sample of testers, increasing with every new user. Even though not every one of them spent a part of their time in giving feedback to us, we still got a considerable amount of it in the process, aiding us to make it a better tool.

David Serrano Arce:

My contribution to this work consisted in a series of tasks, explained below:

Development of the core modules and data structures in the search engine, such as:

- System for automated classification of the results (MatchInfo): the classification of suspicious URLs based on the number of sentences matched. It is implemented as a thread-safe set of pairs automatically created based on the given URLs, and a ranking sorted by number of matches.

- Interfaces and engine for searching with any kind of searcher implemented: in order to allow the implementation of multiple web search engines, it was necessary to abstract the searchers. They may implement any way of searching, as long as the have a start method and another one to export their results via a MatchInfo.

- Multi-thread navigation of the best results: direct navigation (without proxies, even if they are enabled) of a given number of best results from the ranking of MatchInfo, in order to get the selection of the most accurate matches.

- Monitor to control the numbers of threads and connections: One of the downsides of this program is its aggressiveness in terms of bandwidth consumption, even with optical fiber LAN connections. This forced us to create a system to limit the total number of connections,

which turned out to be good for the software's accuracy (proxies are unstable, and having hundreds of connections makes it more difficult to detect which one is failing).

- Proxy manager with punctuation system: to solve the main problem of getting blocked by web searchers, it was necessary to use proxies(in large quantities). This required organization to use the best one available each time (least used and still working) from the proxy list. ProxyManager provides a proxy any time is asked for it, the one with less points than any other. Proxies obtain points each time they are used, and a lot of points (penalization) when they fail resolving a query.

- Document and Selection system for the analysis of text files: reading *.txt* files, and text in general, must be followed by analysis. Document extracts the useful information (deleting spaces, newlines, tabulation and other format and style elements) and classifies them in sentences (lists of words) based on a given criteria. Selection provides the most significant sentences for querying, based on filters, the longest sentences for example. Its main task is to filter the most relevant information from the text, but also to reduce the intensive use of the network.

Adapting external libraries or resources:

- PDF parsing: using the PDFBox library, any document *printable* to PDF, can be converted to a Document. PDF format adds more difficulties, sometimes the text is separated in a strange way, or populated with strange characters. All of that is filtered in order to get readable text.

- Coding of the Yahoo queries and its results parser: this was also the first implementation of a searcher, which also served as a template for the Google searcher, in terms of calling and managing as much threads as queries, waiting for them to finish and stopping them if necessary.

- Getting fresh proxies from the web: having a static list of proxies would not solve the problem of getting blocked by search engines. Sooner or later, they would be banned, even more

if they are used simultaneously by multiple users. So, this program needed *fresh* proxies. There are web pages that offer a list of those proxies for free, and so this part of the code just reads their newest entries. It also keeps searches for more proxies any minute (the more time the user spends searching, the more stressed the current proxies get).