International Journal
Information Technology Applications (ITA)

Volume 12, Number 2, December 2023

COMPARISON OF DOCUMENT SIMILARITIES USING FREQUENCY METHODS

Ján Cigánek, Filip Žemla

Abstract:

This paper analyses the current state of plagiarism. As a result of this work, the proposed tool is implemented and the accuracy of the solution is verified. The analysis and design address preprocessing techniques, file conversion to different formats, algorithms aimed at comparing document similarity and graphical techniques to represent these similarities. Article focuses in particular on frequency methods and graphical representation called Tag Cloud.

Keywords:

Plagiarism, detection, word frequency, tag cloud.

Introduction

Nowadays, plagiarism can be explained as the theft of thoughts and ideas, which the plagiarist later presents as his own. Plagiarism is divided into intentional and unintentional. However, both possibilities are unacceptable in an academic environment. The biggest threat that plagiarism poses to us is the fact that it is no longer just a last resort for students, but is also a common practice in colleges and universities. In this way, it creates a problem that serves to increase the disincentive to further study. It is almost impossible to accurately detect plagiarism when there are many students in particular subjects in schools. Therefore, it was decided to develop a plagiarism comparison tool to compare similarities in text documents. Existing methods will be compared and their suitability for use will be evaluated. Systems used by the public for similarity detection will be used and their effectiveness when used as a basis for the proposed implementation will be assessed. There are many anti-plagiarism tools, some of which are described in detail in Section 1, but they are mostly too difficult to be used in a standard classroom setting. Our target is to develop a simplified tool that would be challenging to fool. Thus, such a tool should not require too much time to refract, which would disqualify it against regular study of the topic under consideration. The tool should be used for the purpose of detecting plagiarism in subjects taught at Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava (FEI STU).

1 Analysis

Within our research we analyzed 2 existing applications used for document authentication at FEI STU. **PlaDeS** is a freely available application for detecting plagiarism in text documents created by students of FEI STU [1]. (Fig.1) shows the basic architecture of PlaDeS tool. It uses the 3-gram method for benchmarking. Acceleration is achieved by using parallelism to process pairs in separate threads and setting the similarity threshold above 0% [1].

The application supports .doc/docx, .pdf and .txt files. First, it analyses different files and preprocesses them by lemmatization and removing stop words. Subsequently, it uses the technique of n-grams, LSC, TF-IDF and metadata. The graphic output is a graph of dependencies. It filters similarity results and exports them to .pdf, .csv formats. This tool gives the possibility to save the project and display the statistics of document similarity.

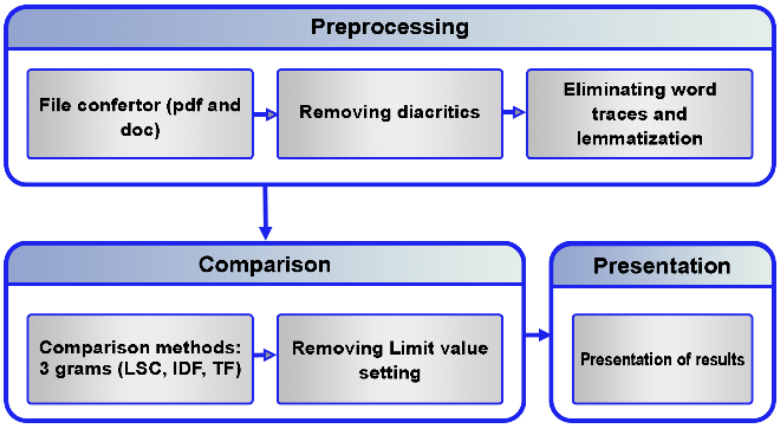

Fig.1. Architecture of PlaDeS.

SimPaD is based on the similarity of words in documents on the web. SimPaD takes into account not only substitution, addition and deletion, but also splitting and joining of sentences based on word similarity. It is used for English texts.

Short sentences and words are removed to achieve higher efficiency. Sentences that have less than 12 words before preprocessing are not taken into account. SimPaD also uses n-gram methods [2].

After a more detailed examination of various methods, metrics and methods for comparing similarities in text documents, we have worked out a possible suitable solution.

Since we want to compare texts in the Slovak language, and SimPaD detection is created for the primary purpose of finding similarities in another language, the basis for implementing the solution is the PlaDeS application.

The implemented method is word frequency. PlaDes application, which uses n-grams, will be used to compare the results. For visualization, the method of tag clouds and verbal notation of percentage similarity are tried [2, 3].

A. Methods for detecting plagiarism

The process by which we achieve the desired result, in our case finding the similarity of text documents, universally consists of three basic parts. These phases are not final and each uses different methods. Their sequence can be seen in Fig.2 [3].

The preprocessing phase includes the acquisition of text from different file formats, their conversion to the same form, and various operations with the given text documents for subsequent comparison purposes. There are several preprocessing factors, including the number of documents being processed simultaneously, the effort to understand the text, and the use of linguistic rules [4].

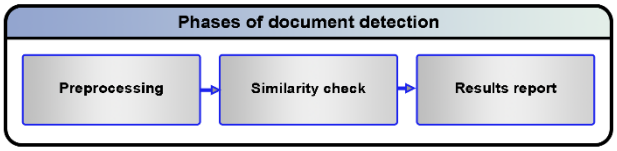

Fig.2. Phases of document similarity detection and their sequence.

The most common ways for preprocessing the text are the removal of stop words, tokenization, lemmatization, stemmatization, normalization, and synonyms replacement. Some of these methods require external resources, such as a dictionary of the relevant language for lemmatization.

The goal of lemmatization is to convert words into their basic form. We denote this shape by the term lemma, which means the "dictionary" shape of the token. It doesn't work for words that aren't in the dictionary, such as names and new colloquialisms. For lemmatization, we will use a database of words in a non-basic form, where a lemma is assigned to each word.

Stemmatization generalizes, shortens the text. Its basic task is the elimination of suffixes and prefixes. It converts words to their root, stem. In this modification, stemmatization does not consider the meaning of the word, nor the context of the text, and therefore it is not certain whether it will find the real root of the word.

By normalization, we check the replacement by hyperonyms and the replacement by synonyms, as the name suggests, we check the replacement by synonyms. Hyperonyms are words superior in meaning [4, 5].

The similarity check on already pre-processed texts uses the following techniques [4, 5]:

- n-gram technique,
- searching the longest common substring,
- word frequency technique,
- technique of inverse word frequencies,
- latent semantic analysis.

Searching the longest common subsequence (**Longest Common Subsequence**) - with this method, the longest common sequence of two documents is determined. Its advantage is a simple principle. However, it is slow, and for documents of more than ten pages, it starts to lose its functionality.

LSA (**Latent Semantic Analysis**) creates sets based on the contextual meaning of words in a set of documents. It assumes that words that are contextually close to each other will be found in the same parts of the text. It can use a matrix that records the occurrence of words in a document. The rows of the matrix represent terms and the columns documents.

Word frequency technique compares word histograms to produce hashmaps. The probability of similarity is calculated according to the intersection with the specified tolerance of the number of occurrences of words.

The term "**n-gram**" defines a sequence of n consecutive items from a given sequence. The model calculates the probability of occurrence of the last word of the n-gram from the previous n-grams. The method is devoted to the detection of overlapping parts of the document and the elimination of the problem of text displacement. The displacement of the internal part of the text is caused by sentence modification, which is the most common technique for masking the copied text and in most cases prevents the exact comparison of strings.

Part of the similarity check in this analysis is also the evaluation of the percentage similarity. That is, whether the given document is plagiarized or not. The best way is to visualize the results and leave the final decision to the individual user [4, 5, 6].

So, the last phase of similarity detection is the presentation of the results. The calculation of similarity proceeds according to the used comparison method: the number of common tokens, the most common n-grams, and a combination of several techniques.

Existing visualization methods are [3, 4]:

- graph of the connection of similar documents,
- pie chart of connection,
- color histogram,
- arc diagram,
- side-by-side,
- tag clouds,
- word spectrum diagram.

In our proposal, a decision is made between using side-by-side, which means color-coded similar blocks side by side, or tag clouds. We can see these two methods shown in (Fig.3) and (Fig.4). Tag clouds contain a list of words in a given document, and the words have a defined size based on their occurrence in the document. Words that appear in both documents are placed in the center or at the edge if they appear in only one document.

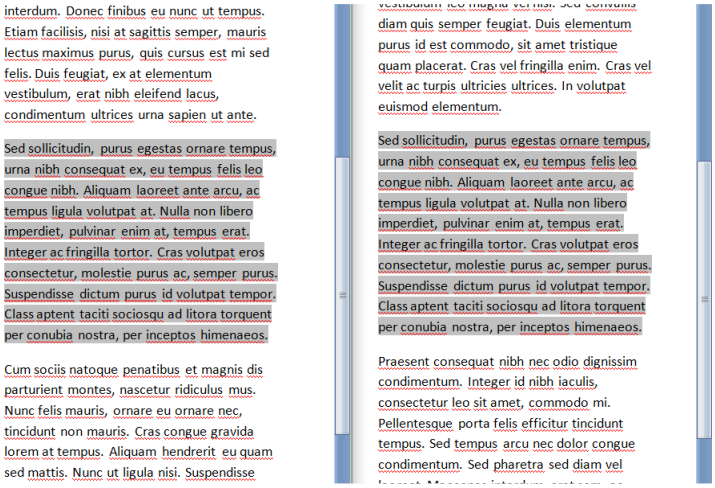

Fig.3. Visualization of similarity results using the side-by-side method.

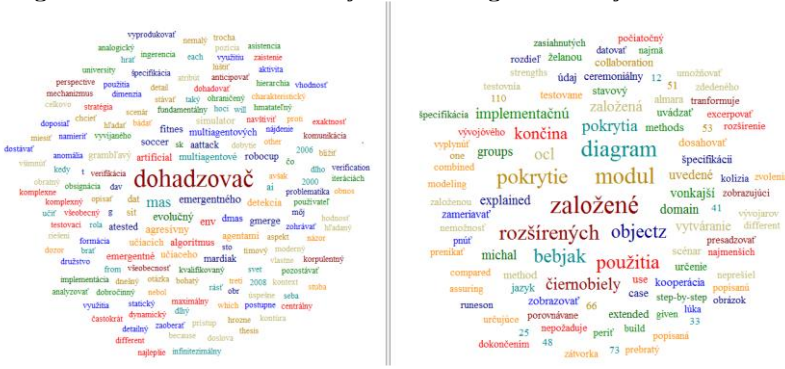

Fig.4. Visualization of similarity results using the tag clouds method.

For the design and subsequent implementation of the prototype, it is first necessary to specify the functional and non-functional requirements. Functionally, they speak directly about the functionality of the system, what it should do. Non-functional requirements are those that do not relate to what the system is supposed to do. They typically talk about system parameters.

Functional requirements for the system are the following.

- Retrieving text and loading it into the system is possible from different types of files.
- Running the loaded texts through tokenization and filtering.
- Individual words of the read text are lemmatized.
- Individual words of the loaded text are synonymized.
- Creation of a corpus of documents.
- The system will allow the user to compare documents.
- The user has the option to choose between one or more comparison methods.
- The similarity search algorithm does not have a long lifetime.
- Visualization of the resulting word vectors using the tag clouds or side-by-side method.

We added the use case of the system to the functional requirements. We can see it in (Fig.5).

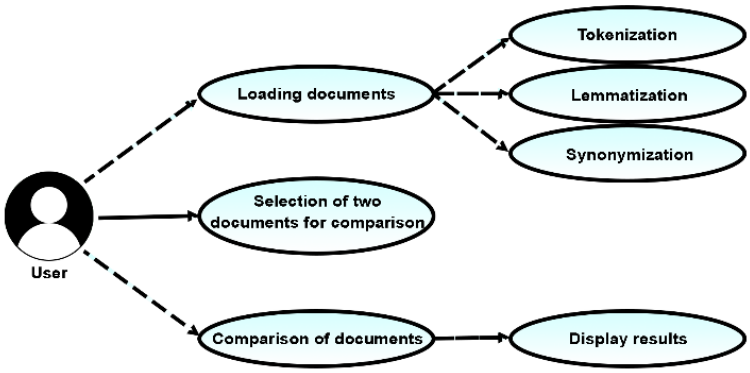

Fig.5. Use case of the system.

On the contrary, the non-functional requirements are:

- Minimization of memory requirements.
- The application implements one or more comparison algorithms.
- The system's graphic interface is user-friendly and easy to operate.
- Similarity matching algorithm uses word frequency technique.
- The system is compatible with the Windows operating system.

2 Design of the Tool

During the implementation, we decided to use the C# programming language with the Visual Studio programming environment. For preprocessing, namely text extraction from the file, we decided to use AbiWord and iTextSharp.

AbiWord is a freely available program, similar to Microsoft Word. It is cross-platform, it has dictionaries for more than 30 languages. It requires to have at least Windows version 2000 and above and 16MB of RAM.

iTextSharp is a PDF library that allows to create, adapt and preserve PDF documents. iText is used in Java, .NET and Android. iTextSharp is portable to .NET [7, 8].

To better describe our application, we show the system architecture in (Fig.6).

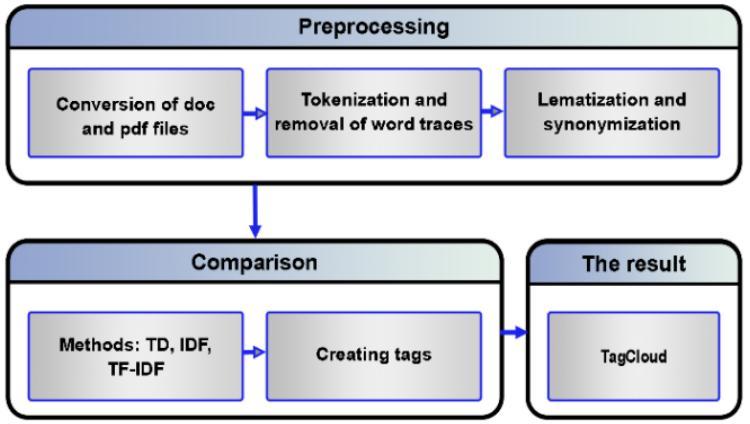

Fig.6. System architecture.

It is important that the environment of even the most demanding applications is simple and intuitive. We also adapted the user interface of our application as shown in (Fig.7).

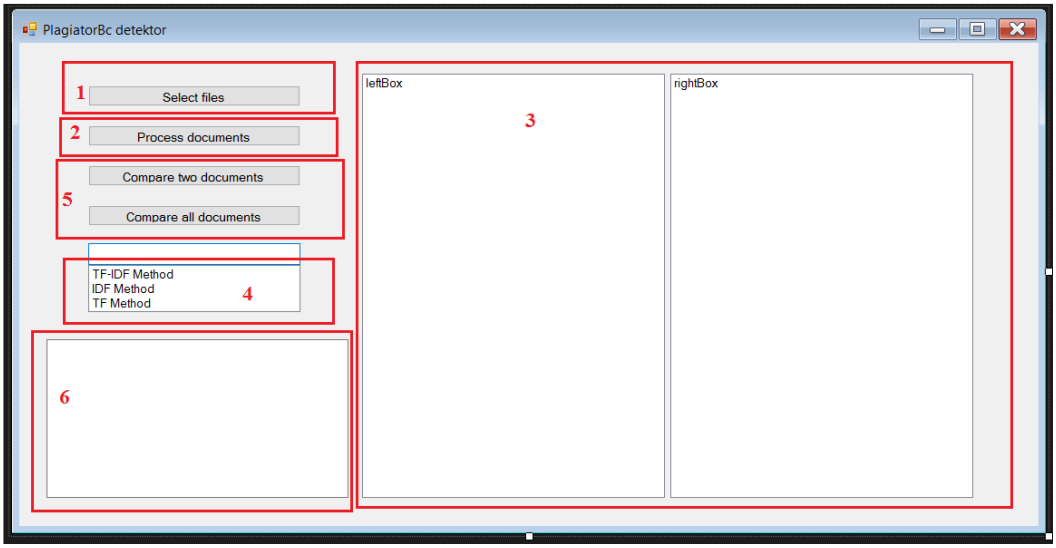

Fig.7. GUI of the system.

We have marked the image with numbers for better understanding. After starting the application, the user first selects the files (1) that he wants to compare. After adding them, he chooses document processing (2), after which they appear in two ListBoxes (3). For the calculation, it is possible to choose a method using ComboBox (4). For the subsequent calculation, he presses Button (5). The user has two Buttons to choose from. One compares two documents using Tag Clouds, and the other lists the percentage match of all documents in the corpus. We write all the information, that we want the user to have at hand, in the ListBox (6).

A. Document loading, conversion and preprocessing

When loading and converting documents, we implemented the IParser interface, which is followed by the Tokenizer class. IParser is used to convert *.doc and *.pdf format files. It is therefore used by two classes, namely ParserDoc and ParserPdf.

ParserDoc uses the external program AbiWord to convert a *.doc file into a plain text file with UTF-8 encoding. The text file is then read using the StreamReader class and processed into a string using the StringBuilder.

ParserPdf works with the external library ItextSharp.dll, which allows to retrieve strings on specific pages of a PDF file. We then merge the obtained strings into one single string using StringBuilder [8].

Tokenizer uses C# regular expressions for tokenization. Any string consisting of meaningful characters is considered a word. Among the meaningful signs we consider dot and comma. StopWords is a class representing the removal of words the listed in the stop list. The list of stop words is stored in a collection of type List<>. This list is read from a file where a line represents one stop word. The verified word must be recorded in lowercase letters.

When implementing lemmatization, we used the dictionary method. The dictionary was obtained from the work in CDB format. Using the procedure, we converted this dictionary to plain text with UTF-8 encoding. The method is implemented in the LematizerSK class. For faster lemmatization, the HashTable class is used. The dictionary is stored by lines. As with lemmatization, we also used the dictionary for synonymization in the Synonymizer class. An already existing dictionary was used, from which a text file was created. HashTable is also used here to speed up the synonymization process [1, 8, 10].

B. Determining the similarity of documents

To implement the visualization page of our application for the selected method - Tag Clouds, we used an external library. For integration into VS 2019, in which we programmed, we wrote an adapter for the given control.

The Tag Clouds method is displayed in the case when the user wants to compare only two specific documents. Each document has its own tag cloud created. The tag cloud is populated with information from the TagCloudData class, which uses TagWord. The TagWord class contains information about a specific document term: its weight, a list of other documents in the corpus where it is found, and the term itself. These TagWords form two sheets, one for each document. In addition, TagCloudData contains the names of the files from which the information was drawn and the percentage dependence of one document on another using the method implemented by us, described below. We also use this method if the user wants to compare all documents in the corpus at once [9].

For the determined similarities of the pre-processed documents, we used the Term Frequency - Inverse Dense Frequency methods. In addition, a simple method was added, which was used to calculate the percentage of documents matching and as a reference when testing other methods.

Our implemented method is based on the normalization of vectors that tell us how far the documents are from each other. After adding them up, we get a cumulative error that expresses the degree of dissimilarity. To obtain the overall similarity of the documents in percent, we subtract this absolute error from unity and multiply by one hundred. The resulting sum indicates the dependence of the given document on the pivot.

The normalization of a vector whose magnitude must be one is performed as follows. For each TagWord that expresses a term in the document, we count its frequency in the given document and divide by the number of all terms in the given document. In this way, we get the value of the normalized vector, which we subsequently use in formula (1). To reduce the complexity of the cycle, we remove all terms whose weight is equal to zero.

For each TagWord in the pivot document, we calculate the absolute value of the difference of the normalized vectors of two identical terms. The first of the vectors belongs to the term of the pivot document, and the second to the vector of the second document. We are looking for the document dependent on the pivot.

$$CumulativeError = \Sigma abs(PivotWordVector - DocumentWordVector) \quad (1)$$

$$Result = [1 - (CumulativeError)] * 100 \quad (2)$$

3 Verification of the Correctness of the Solution

We selected sixteen documents for the first corpus. In this sample, there are known pairs of plagiarisms on which we could verify the size of the estimated similarity. For the existing solution, we selected two comparison methods from an external application - PlaDes: Q-grams and TF-IDF. Q-grams are a modified 3-gram document comparison method.

We compared these two methods with our simple evaluation of two documents. The goal of this experiment is to find out whether the proposed method is close to the other two, and whether it is appropriate to continue using it to express the percentage similarity of two documents in our application.

Table 1. Experiment 1. average deviations

Method	Average deviations
Simple Method / Q-gram	8.16 %
Simple Method / TF-IDF	5.06 %

From (Tab.1) and (Fig.8), we can see that our chosen method has only a small deviation against the other two methods. To prove that we don't need a smaller deviation, we will point out that our program also has a visual side to compare two documents - tag clouds. These Tag clouds are created using TF, IDF or TF-IDF methods. As an example, we will show the real result of our prototype, where we will compare two documents that had the highest deviation when comparing Simple Method and TF-IDF.

The TF-IDF method, unlike the Simple Method, calculates the resulting value of term weights based on their frequency in several documents. We also took this fact into account in Tag Cloud. If the user wants to find out how many other documents from the corpus a given term is found in, it is enough to click on the given term in the Tag Cloud. A window with the required information will appear, as we see in (Fig.9).

For the second corpus, we selected the same documents, but compared them to each other, not only known pairs of plagiarists, but also among themselves. There will be comparatively more results than in the first experiment and thus the final result will be more objective. The goal, similar to the first experiment, is to find out credibility of the method.

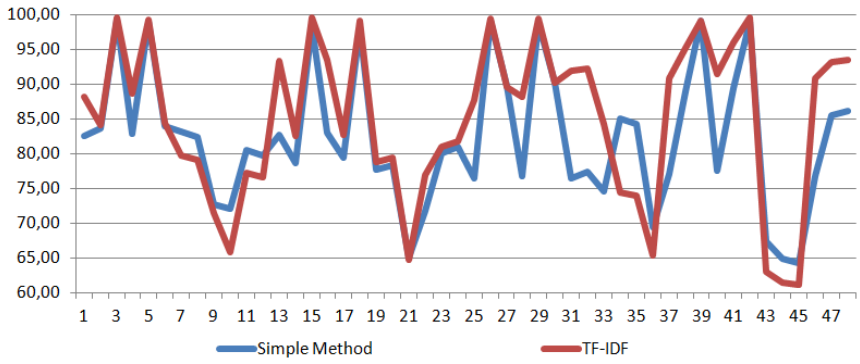

Fig.8. Similarity of documents (Simple Method – TF-IDF).

Fig.9. Graphical results in application.

Table 2. Experiment 2. table with statistics

Method	Statistics	Value
Simple Method / TF-IDF	Average deviation	18.07 %
	Number of comparisons with deviation 0-10 %	53
	Number of comparisons with deviation 10-20 %	87
	Number of comparisons with deviation 20-30 %	99
Simple Method / Q-gram	Average deviation	16.04 %
	Number of comparisons with deviation 0-10 %	40
	Number of comparisons with deviation 10-20 %	68
	Number of comparisons with deviation 20-30 %	132

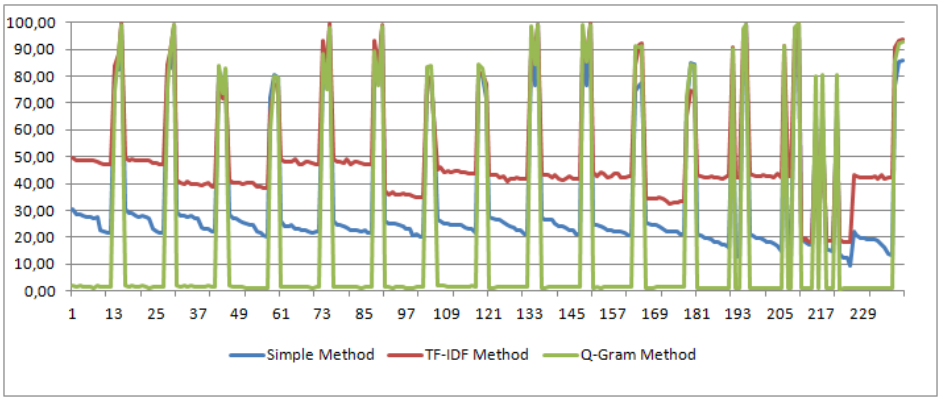

Fig.10. Similarity of documents for different methods.

In (Fig.10) we can see the similarity graph for the individual methods that were compared in this experiment. As you can see, our Simple Method has no problem comparing a high percentage of similarity and thus alerting to an attempted plagiarism. The difference seen for 50 percent or less agreement in methods can be attributed to the fact that these were papers from one problem area - Software Engineering Essays.

Conclusion

The main goal of the work was to supply an existing application for comparing texts and the corpus of submitted student works.

We compared the proposed preprocessing method and the implemented comparison methods with existing methods.

After verifying the correctness of the simple method, it is possible to claim that the used comparison methods are appropriately chosen and the presence layer was appropriately chosen, which will allow the user to display and manipulate the detected similarity results.

Identified problems of the prototype during the implementation and during experimentation, a few shortcomings of the prototype were revealed. Some were removed in time, but it will be continue to try to do so for the following. The visual page of the results of the comparison of all documents in the corpus is not acceptable for the simple and intuitive graphical interface. A possible solution is to find another display method where the results will be easy to read and the user could sort them according to his own will. The time complexity of the process of comparing all documents is too lengthy. 10 comparisons take place in one millisecond. To compare 93 documents with each other during experimentation, the elapsed time to obtain the result was 14 minutes. File formats allowed for comparison are pdf and doc. There is also a possible expansion for other formats.

In the future, the system could be modified to use multiple cores in the computer, or to implement the work of the algorithm in multiple threads. It would be advisable to sort the results or add the option of searching for a specific document in the results. At the same time, it would be good to add the possibility of exporting the results to a file.

Finally, there was implemented only one method of graphical representation of similarity in the paper. In future work, these methods could be expanded, and the user could be given the choice to display the result using, for example, the side-by-side method, or a graph of the connection of similar documents.

Acknowledgement

This paper was supported by the Slovak Grant Agency VEGA 1/0637/23 and KEGA 010STU-4/2023, and by the Scientific Grant APVV-21-0125.

References

[1] D. Chudá, T. Hlatký, M. Kompan, M. Ludvík, R. Švajdlenka. *PlaDes - Plagiarism Detection System*. Bratislava Faculty of Electrical Engineering and Information Technology of STU in Bratislava. [Online]. Available from: http://www2.fiit.stuba.sk/~chuda/plagiarism/PlaDeS.html

[2] Pera, Maria & Ng, Yiu-kai, SimPaD: *A word-similarity sentence-based plagiarism detection tool*, Web documents. Web Intelligence and Agent Systems 9. 27-41. 10.3233/WIA-2011-0203.

[3] D. Chudá. *Advances in database and knowledge technologies 1*. Petr Šaloun, Ján Genci. 1. VSB – Technical University of Ostrava. s. 99. ISBN: 978-80-248-2813-8.

[4] Z. Češka. *Use of modern approaches for plagiarism detection*. [Online] 2008. Available from: http://textmining.zcu.cz/publications/ModerniPristupyProPlagiaty-ITAT2008-Czech.pdf

[5] R. Garabík, L. Gianitsová, A. Horák, M. Šimková. *Tokenization, lemmatization and morphological annotation of the Slovak national corpus*. [Online] 2004. Available from: http://korpus.sk/attachments/publications/2004-garabikgianitsova-horak-simkova-tokenizacia.pdf

[6] M. Pataki. *Plagiarism detection and document chunking methods*. Proceedings of the twelfth international conference on World Wide Web, WWW2003. Budapest. NIIF, 2003.

[7] iText Group nv (HQ Belgium). *IText*. [Online]. Available from: https://itextpdf.com/

[8] the AbiSource community. *AbiWorld* [Online]. Available from: https://www.abisource.com/

[9] A. Shvets, G. Frey, and M. Pavlova. *Design patterns*. [Online]. Available from: https://sourcemaking.com/

[10] R. Garabik. *Slovak morphology analyzer based on levenshtein edit operations*, 2006.

Authors

Ing. Ján Cigánek, PhD.
Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology in Bratislava, Slovakia
jan.ciganek@stuba.sk
He was born in 1981 in Malacky, Slovakia. He received the diploma and PhD. degree in Automatic Control from the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology (FEI STU) in Bratislava, in 2005 and 2010, respectively. He is now Assistant Professors at Institute of Automotive Mechatronics FEI STU in Bratislava. His research interests include optimization, robust control design, computational tools, SCADA systems, big data, and hybrid systems.

Ing. Filip Žemla, PhD.
Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology in Bratislava, Slovakia
filip.zemla@icloud.com
He received the diploma and PhD. degree in Automotive Mechatronics from the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology (FEI STU) in Bratislava, in 2020 and 2023, respectively. He is now External colleague at Institute of Automotive Mechatronics FEI STU in Bratislava. The research is oriented to virtualization and optimization of modern manufacture processes. His main skills are SCADA systems, database systems and front-end programming.