Software for Assessing the Performance of Anti-plagiarism Programs

Marius Popescu[1], Csiki Laszlo[1] and Antoanela Naaji[1,a]

[1]*"Vasile Goldis" Western University of Arad, Engineering and Computer Science Department, 86 Rebreanu St., Arad, Romania*

Abstract. Detecting plagiarism is one of the current issues concerning the process of publishing scientific papers. There are numerous anti-plagiarism programs on the market, some of which are free, other proprietary. However, with a few exceptions, they cannot detect changes brought to original texts, but can only identify copy-pasted paragraphs of a certain length (number of words). The software that we developed and which we are presenting in this paper is based on an original algorithm using Java, and is aimed at assessing the performance of anti-plagiarism software. This program shows that such software has vulnerabilities and can be easily sidestepped, especially by programmers. The algorithm's method consists of reading a .docx document, and then replacing every fifth word in the sentence with a synonym existing in text files attached to the software. At the end, when the algorithm completes its cycle, the result is saved in a new .docx document. In order to demonstrate the effectiveness of our program as a tool for assessing the performance of anti-plagiarism software, the paper presents a comparative analysis of 4 such programs, based on the percentages of originality and similarity obtained.

1 Introduction

Since 1980, governments, universities, research institutes and other institutions have been facing an increasing amount of research misconduct. Consequently, policies and procedures have been designed to investigate, adjudicate and prevent such cases. According to the US Office of Research Integrity, misconduct in research means [1]:

"- Fabrication – making up data or results and recording or reporting them;
- Falsification – manipulating research materials, equipment or processes, or changing or omitting data or results, such that the research is not accurately represented in the research record;
- Plagiarism – the appropriation of another person's ideas, processes, results, or words without giving appropriate credit;
- Research misconduct does not include honest error or differences of opinion (45CFR 93.103)" [1].

The new approach to misconduct, which is more effective than attempting to catch and punish, is prevention by encouraging good conduct in research.

As regards plagiarism, it is the appropriation of ideas, methods, procedures, technologies, results or texts of another person, regardless of how they were obtained, and the presentation of such as one's own creation [2]. This is a severe infringement of moral rules in these professional communities in which the originality of creations is acknowledged and rewarded.

Generally speaking, plagiarism is an infringement of both the author's moral rights and intellectual property rights over their creation. Nowadays, the phenomenon is enhanced by the possibility to access Internet and the ease with which texts, ideas or images can be copied from doctoral theses, research papers, or articles existing online. Although there are numerous previous examples, in the 1980s it can be seen a slow increase in plagiarism, for about 8 years, followed by a sudden rise before 1990, which lasted for almost 10 years. In last years, the number of cases decreased, largely due to the development of new methods for detecting plagiarism, using specialized software [3].

Plagiarism based on texts derived from the Internet is currently called online plagiarism. Due to the increasing number of online publications, the methods of detecting plagiarism should be more diverse. Thus emerged new services and specialized software to detect plagiarism, free or for payment, some directly accessible on various websites, others through computer programs. In general, websites and programs can detect copying-and-pasting from texts found on the internet or in own databases, not considering punctuation (usually inverted commas), but they cannot detect rephrased sentences or translations from foreign languages [4]. Since anti-plagiarism software cannot pronounce itself on the plagiarism of ideas, but only of texts [5], it does not provide high performance.

Linguistic phenomena underlying plagiarism were analyzed only after these systems were designed, and are a key issue in improving such software. Various types of plagiarism were identified [2], such as plagiarism of ideas, references, authorship, word-by-word and paraphrases.

[a] anaaji@uvvg.ro

In the first case, ideas, knowledge or theories are reclaimed without appropriate citation. Reference and authorship plagiarism includes entire citations or documents, with no mention of authorship. Word-by-word plagiarism is known as copy-paste, or textual copy, and consists of the exact copy of a text (fragment) from a source in the plagiarized document. As regards paraphrase plagiarism, it is often used to conceal the act of plagiarism, and expresses the same content under a different form. Paraphrasing is generally defined as invariability between different formulations and is the linguistic mechanism underlying many acts of plagiarism. [6].

The paper presents a software application, based on an original algorithm, which modifies a certain text by automatically replacing words with their synonyms. Its primary role is to detect vulnerability in anti-plagiarism software, as well as to evaluate its performance. As method, the original text is checked by four anti-plagiarism programs, three of them free and one proprietary. Subsequently, they can be modified using our program (by replacing words with their synonyms), and then proceed to a new check.

The following sections present the algorithm and how it is implemented, its testing on the four anti-plagiarism programs, the discussions on results, and conclusions.

2 Description and implementation of the algorithm

The program was created in Java and has two classes, namely, *DocumentHandle* and *App*. After reading the paragraphs in a .docx file, each fifth word is replaced with a synonym in text files [7], and if no synonym is found, then it proceeds to the following word, until it reaches the end of the document. The program ends, and the result is saved in a new docx document. The most important of the two classes is the *DocumentHandle* class, which manages the entire algorithm. This is where variables are declared (`String inputFilePath, String outputFilePath, XWPFDocumentdoc`) and implemented (*DocumentHandle* class requires 2 parameters: `String inputFilePath` and `String outputFilePath`), while also establishing the necessary methods (`String smallInput(), String get_sin_file(), String getFirstLetter(), String changeWord(), void rebuildDocument()` and `StringBuilder mainEngine()`). Thus, the `smallInput` method reads a parameter which is the word *inputWord*.

```
private String smallInput(String fullWord){
 String smallInput=null;
 if(fullWord.length()<4){smallInput=fullWord;}
 else
if((fullWord.length()>3)&&(fullWord.length()<6)){
smallInput=fullWord.substring(0,fullWord.length()-
1);} else
if((fullWord.length()>5)&&(fullWord.length()<8)){
smallInput=fullWord.substring(0,fullWord.length()-
2);} else if(fullWord.length()>=8){
smallInput=fullWord.substring(0,fullWord.length()-
2);} return smallInput;}
```

For a more accurate search, the methods checks the length of the word and if it is longer than three, then the last (one or two) letters are deleted.

```
private String get_sin_file(String letter){
  String letterFilePath=null; File dir=new
File("C:/Desktop/Plagiarism/synonyms/");
 FileFilter fileFilter=new
WildcardFileFilter("*_"+letter+".txt");
 File[] files=dir.listFiles(fileFilter);
 for (int i=0;i<files.length;i++){
letterFilePath=files[i].getAbsolutePath();}
 if(letterFilePath==null) {letterFilePath=
"C:/Desktop/Plagiarism/synonyms/litera_a.txt";}
 return letterFilePath;}
```

The method `get_sin_file()` reads a parameter which is the first letter of the word and then looks for the file containing words beginning with that letter and their synonyms.

```
private String getFirstLetters(String word){
  String firstLetters="";
  word=word.replaceAll("[.,-]","");
  if(word.equals("")){return null;}
  else{for(String s:word.split(" "))
  {firstLetters+=s.charAt(0);}}
  return firstLetters;}
```

The method `getFirstLetter()` reads a parameter which is the word *inputWord* and returns a *string* which contains the first letter in the word.

```
private String changeWord(String
unformattedInput){String inputWord=
 unformattedInput.replaceAll("[.,”]","");
 String outputWord=null;
 String firstLetter=getFirstLetters(inputWord);
 if(firstLetter!=null){
  String filePath=get_sin_file(firstLetter);
  File file=new File(filePath);
  if (inputWord.length()<3){return inputWord;}
  else{try{Scanner scanner=new Scanner(file);
  int lineNum=0;
  while (scanner.hasNextLine()){
  String lineFromFile=scanner.nextLine();
  String[] splitLine=lineFromFile.split(" ");
if (splitLine[0].contains(smallInput(inputWord)))
{Stringline=lineFromFile;outputWord="["+inputWord
  +">"+line.substring(line.indexOf(":")+2)+"]";
  return outputWord;}}
  if (outputWord==null){outputWord="[Change this
      ->"+inputWord+"]";}}
  catch (FileNotFoundException e){
       e.printStackTrace();}}}
  else{outputWord="[Change this-
>"+inputWord+"]";} return outputWord;}
```

Likewise, the `changeWord()` method reads a parameter which is a *inputWord* and then runs the method `getFirstLetter()` and the method `get_sin_file()`. Then, the word is changed if it exists in that file, and the synonym is returned. If the word does not exist in the list, then a square bracket is returned containing the following line: `[Change this->inputWord]`.

```
private void rebuildDocument(){try{
   this.doc.write(new
FileOutputStream(this.outputFilePath));
   System.out.println("File saved to:"+
```

```
this.outputFilePath);}
   catch(IOException e){e.printStackTrace();}}
```

The `rebuildDocument()` method recreates the formatted .docx document and saves it.

```
public StringBuilder mainEngine(){
  StringBuilder returnString=new StringBuilder();
  int counter=1;
  for(XWPFParagraph p:this.doc.getParagraphs()){
  List<XWPFRun> runs=p.getRuns();
  if(runs!=null){for (XWPFRun r:runs){
    String text=r.getText(0);
    if(text!=null) {//find the fifth words
    String[] split=text.split(" ");
    for (int i=4;i<split.length;i=i+5){
    System.out.print("original world:"+split[i]);
    String currentWord=split[i];
    returnString.append(counter+".Original
world:"+currentWord);returnString.append("changed
to:"+changeWord(currentWord)+"\n");
text=text.replace(currentWord,changeWord(currentWor
d));counter++;}
   r.setText(text,0);}}}}
```

The `mainEngine()` method is the engine of the entire algorithm, where all major events occur. Firstly, we take each paragraph and look for the *n*-th word, which will be replaced. In the end, when no paragraph is left, we call on the method `rebuildDocument()`, which ends the algorithm and returns the result; this can be seen on the interface of the software. The *Synonym file* contains the synonyms of all words in the Romanian language. The algorithm flow is illustrated in a logical schema (Fig.1).

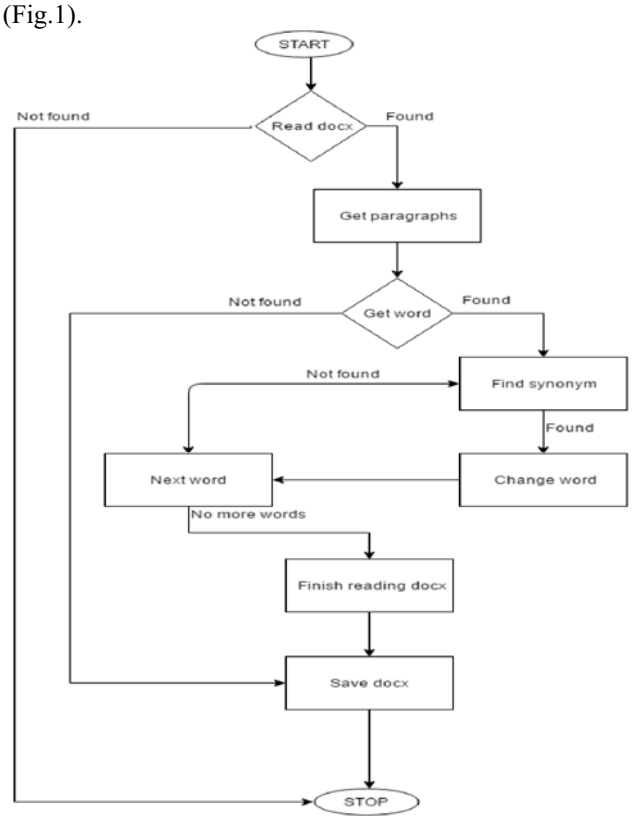


Figure 1. Simplified organization chart of *the Software for Assessing the Performance of Anti-plagiarism Programs*.

In the first step the *input* file, meaning the document which is meant to be modified, is read. In the second step the existence of the document is checked. If it does not exist then the program is stopped with no error. If it does exist, then the most complex part comes along, meaning opening and reading a paragraph, which means – for example – one line, and from there takes each *n*-th word which is *inputWord*. Then, the first letter of the word is taken and that particular file is open, containing synonyms – for each letter there is a separate file which contains the synonyms. For example, *letter_a.txt* which contains the words beginning with the letter *"a"*. After this stage the word is searched in the file and if it does not exist, then it moves to the following word in the paragraph. If it does exist, then the *inputWord* is changed with the synonym. Going further, the following word is taken from the paragraph and the loop is run again until no word remains unevaluated in the paragraph. Then, the following paragraph is taken and the loop is started again. This flow is reprised until the paragraph is finalized. If there is no paragraph left, then the program saves the file and the algorithm is stopped. The program uses the libraries *.apache.poi.xwpf* and *org.apache.commons*; among the most important was the *org.apache.poi.xwpf* library. The program works with this open-source API and can manage a *.docx* document. The software is written under a *Maven* structure, therefore the application can run without error on other *compilers*.

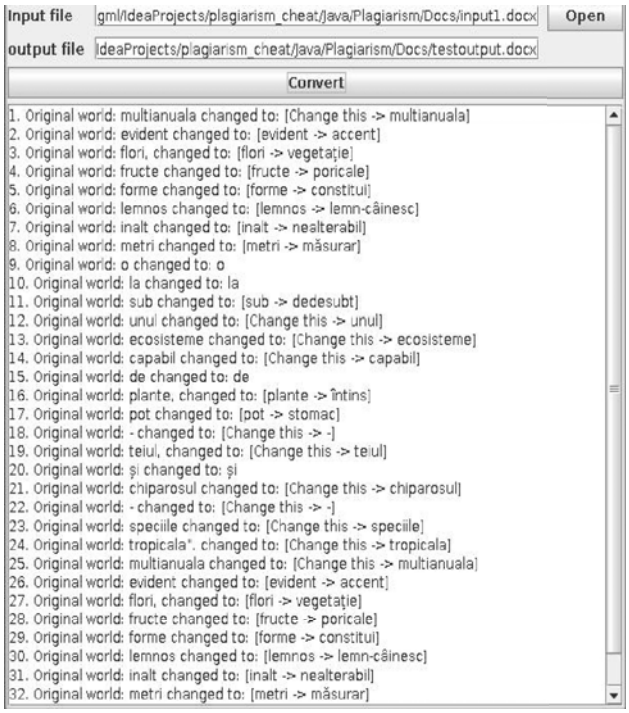


Figure 2. The primary interface of the *Software for Assessing the Performance of Anti-plagiarism programs.*

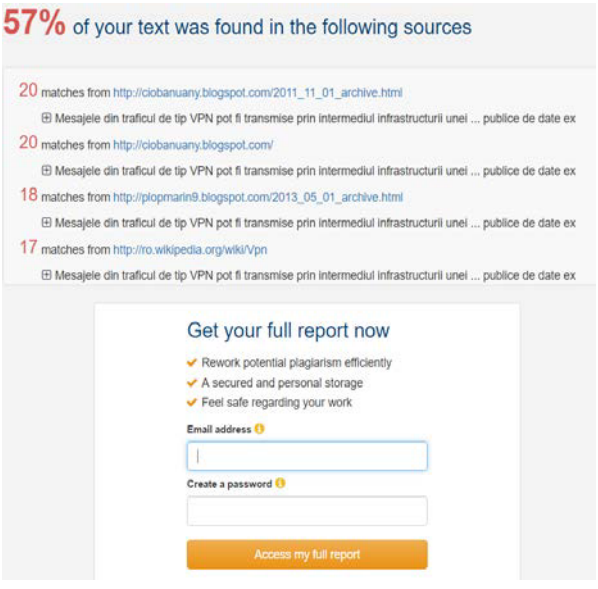
The graphic interface (GUI) of the *Software for Assessing the Performance of Anti-plagiarism programs* is user-friendly (Fig.2). The *Convert* button starts the engine in the background. The desired file is placed in the *input file*, specifying the path/location where it is found in the computer, and the *output file* specifies the location where to save the new document. The program runs and changes words that are found in the dictionary, and those which are not found are placed between braces.

3 Testing the program and discussions on results

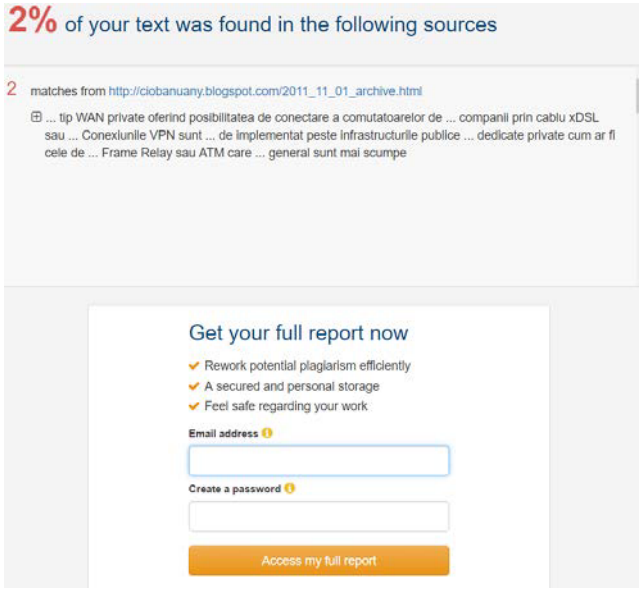
The *Software for Assessing the Performance of Anti-plagiarism programs* is based on an original algorithm, being an open-source program, available on Github, at: https://github.com/laszlocsiki/plagiarism_cheat/tree/development/Java/Plagiarism/src/main/java/plagiarism. To test the performance of the software, a plagiarized document was checked using several online anti-plagiarism websites and a proprietary anti-plagiarism software application.

It is worth mentioning that modifying a document using the algorithm presented only occurs in the case of copy-paste plagiarism. Submitting a plagiarized document to the *Software for Assessing the Performance of Anti-plagiarism programs* does not eliminate plagiarism, but results in a paraphrase plagiarism, which conceals the act of plagiarism, while the document expresses the same content under a different form. The syntactic correctness of the text submitted to the algorithm depends on the synonyms imported from the *Synonyms* file. The new text requires a review or grammatical proofreading, even if it proceeds to check the existence of synonyms for the words found in the sentence on positions "x+1", "x+2" etc., when no synonym is found for the word at position "x", with $x \in [4 \div 9]$. Otherwise, the content of sentences can be distorted or unintelligible.

To begin with, we selected and ran the document in the free plagiarism detection software called *PlagScan* [8], which proved to be a very precise light-weight plagiarism tester. After scanning the plagiarized document, the result was 57% plagiarized, which means that more than half of the content was plagiarized (Fig.3a). After running the document through the *Software for Assessing the Performance of Anti-plagiarism programs*, the result obtained is more than satisfactory, namely, a 2% similarity with the original document (Fig.3b).



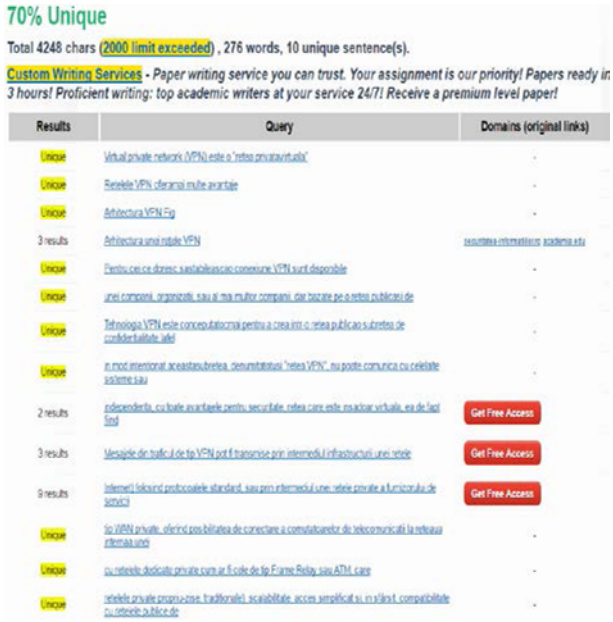
a)



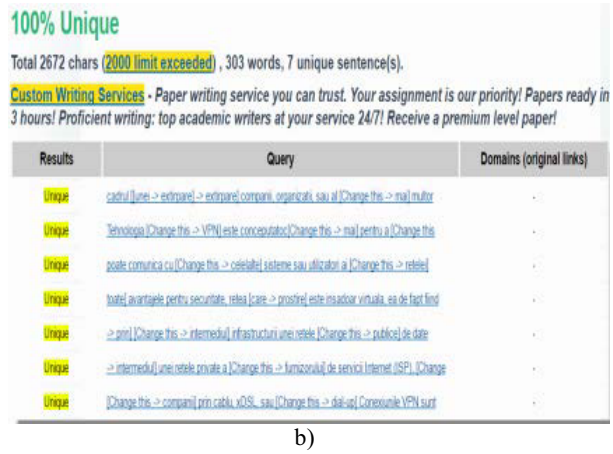
b)

Figure 3. Interface of the *PlagScan* software [8]: a) before running the document through the *Software for Assessing the Performance of Anti-plagiarism programs* b) after running the document through the *Software for Assessing the Performance of Anti-plagiarism programs.*

With the result returned for the chosen document, the program was ran and checked again for two other free applications used to check plagiarism, called *Plagiarism Checker* [9] and *Plagiarisma* [10]. The difference is illustrated in Figures 4 and 5. After the first scanning of the original document with the *Plagiarism Checker* software (Fig.4a), the result was 70% original, meaning that only 30% of the content was copied from other sources. After running the *Software for Assessing the Performance of Anti-plagiarism programs,* the similarity coefficient was null (Fig.4b), meaning that 100% of the text was unique/original, no information was copied from other sources.



a)

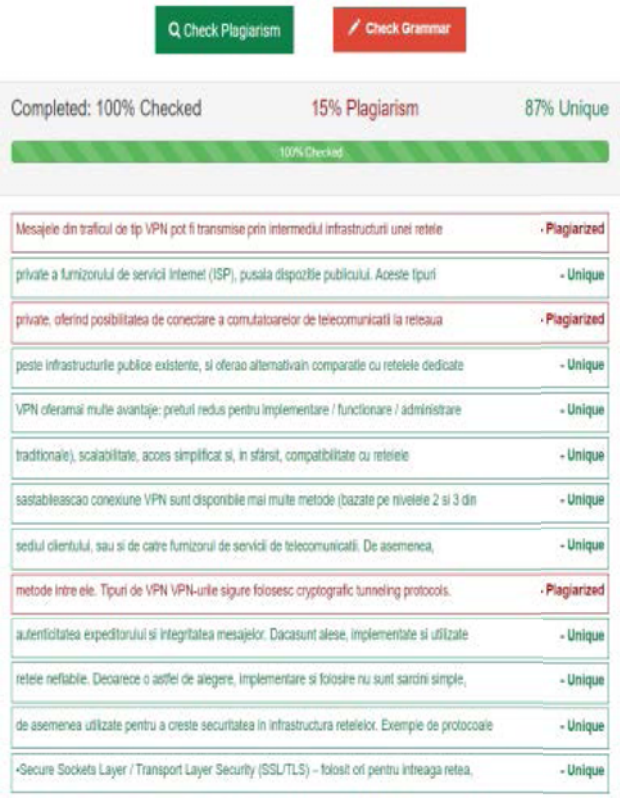


b)

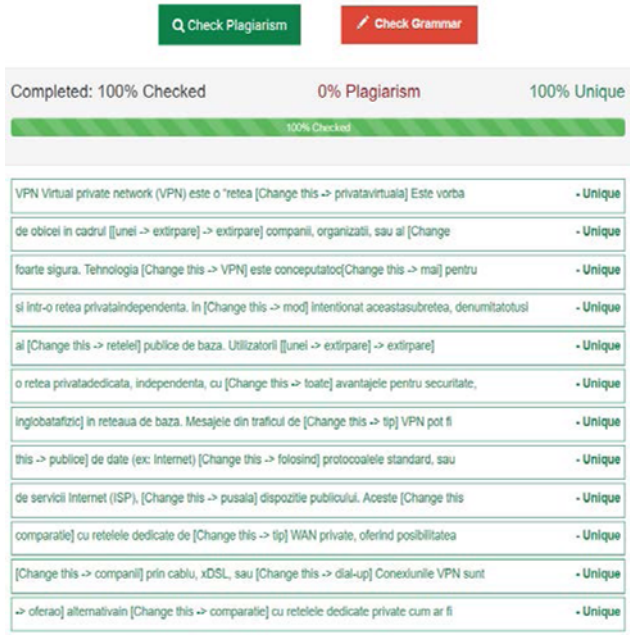
Figure 4. Interface of the *Plagiarism Checker* Program [9]: a) before running the document through *Software for Assessing the Performance of Anti-plagiarism programs* b) after running the document through *Software for Assessing the Performance of Anti-plagiarism programs.*

Small SEO Tools is a tester which has many services connected to SEO, such as *Plagiarism Checker, Keyword Position Checker, Grammar Checker, Spell Checker,* etc. By checking the document with *Plagiarisma*, it initially found that 87% of the content was unique and 15% was plagiarized, meaning from other sources (Fig.5a).

The *Software for Assessing the Performance of Anti-plagiarism programs* was run, then the document was submitted again to the anti-plagiarism software and we can see that the new document is no longer plagiarized (Fig.5b). In other words, the anti-plagiarism software detects a 15% similarity, and after using our software, the anti-plagiarism software established the document to be unique/original.



a)

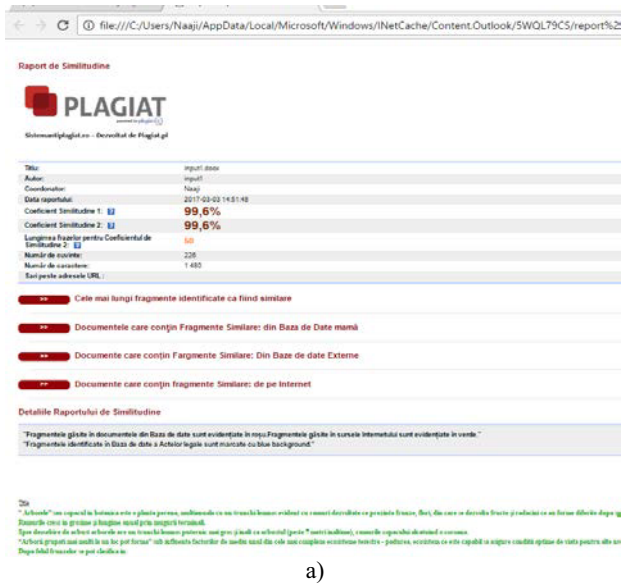


b)

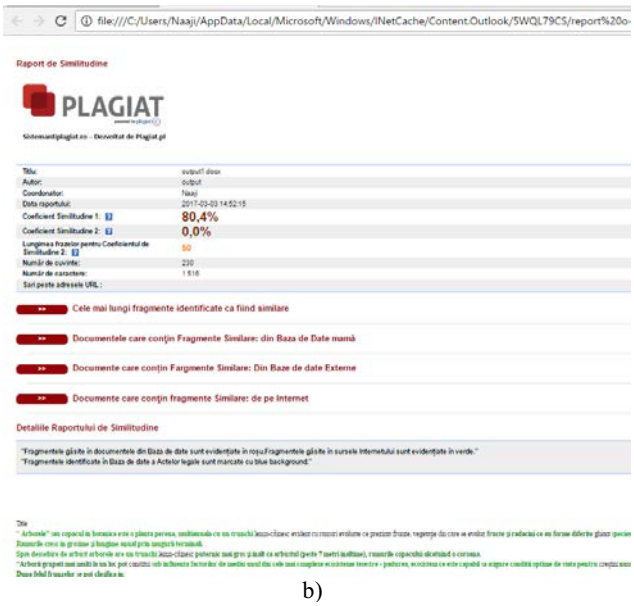
Figure 5. Interface of *Plagiarisma* [10]: a) before running the document through *Software for Assessing the Performance of Anti-plagiarism programs*; b) after running the document through *Software for Assessing the Performance of Anti-plagiarism programs*.

As followed we analyzed the performance of the software by testing the plagiarized document checked with the help of a high-performance anti-plagiarism software application, *Sistem antiplagiat (Anti-plagiarism System)* [11], which evaluates the degree of originality of the text for the document. Details of the Similarity Report refer to concrete aspects concerning plagiarism, such as: fragments found in documents within the database are highlighted in red, those found in Internet sources are colored in green, and those identified in databases of legal documents are marked in a blue background.

After the first scanning of the original document with the *Sistem antiplagiat* software (Fig.6a), the result was 0.1% original, meaning that almost 100% of the content was copied from other sources.



a)



b)

Figure 6. Report generated by the *Sistem antiplagiat* software [11]: a) before running the document through *Software for Assessing the Performance of Anti-plagiarism programs*; b) after running the document through *Software for Assessing the Performance of Anti-plagiarism programs*.

After running the *Software for Assessing the Performance of Anti-plagiarism programs,* the similarity coefficient was still 80.04% (Fig.6b), meaning that only about 20% of the text was unique/original.

For this software, it can be noted that the application calculated 2 similarity coefficients, but the first coefficient is not sufficient, given the small decrease after applying the *Software for Assessing the Performance of Anti-plagiarism programs* (Fig.7). When opting for x=4, changing every fourth word, then the similarity coefficients are improved significantly. Sources marked in bold letters contain fragments which can be plagiarized and exceed the limit of the *Similarity Coefficient 2* in length [11].

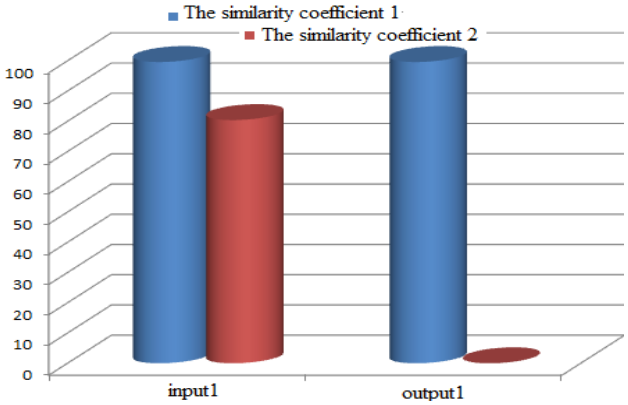


Figure 7. Comparison of results before and after running the document through the *Software for Assessing the Performance of Anti-plagiarism programs*.

A comparative analysis of results obtained by using the free anti-plagiarism programs demonstrates that most of these programs can be "fooled" especially by programmers or even by authors which are not IT specialists, through the simple replacement of words with their synonyms.

Likewise, by performing an analysis of performance on the anti-plagiarism software mentioned above, it can be noted that *Plagiarisma* is not as precise/efficient as *PlagScan*, but it is enough to test plagiarized documents (Fig.8). The explanation of the difference between the results of anti-plagiarism programs resides in the fact that each program uses another method to check the content of the document. For example, one method takes 10 words from a statement and checks that paragraph in other existing sources, and the other method takes only 5 words per statement. For this reason there is a very large difference between anti-plagiarism programs. The users raise the question of using the most precise anti-plagiarism software. Running the *Software for Assessing the Performance of Anti-plagiarism programs* shows that the program which checks 5 words in a paragraph is much more precise than a program checking 10 words in a paragraph.

Thus, results obtained indicate the fact that the *PlagScan* program is more precise, noting that after running *Software for Assessing the Performance of Anti-plagiarism programs*, only 2% of the content was plagiarized.

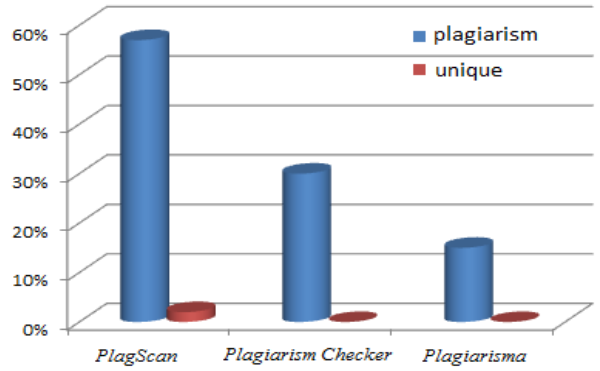


Figure 8. Comparing the results obtained using three anti-plagiarism programs.

Tests conducted previously illustrate the fact that plagiarism testers can be sidestepped, with a simple algorithm and manual review, whereas if the algorithm is more developed, a document can be created to avoid "copy-paste" plagiarism.

The main purpose of *Software for Assessing the Performance of Anti-plagiarism programs* is to compare and detect the most efficient anti-plagiarism software applications.

4 Conclusions

Software for Assessing the Performance of Anti-plagiarism programs is based on an original algorithm, being an open-source program**.** The software was developed only for testing anti-plagiarism programs and it proves that, with a few manual touch-ups, plagiarism tests can be avoided. Any plagiarized document which goes through the algorithm described above may elude detection by anti-plagiarism software, which illustrates that establishing the plagiarism of a document, or ideas

from a certain field of expertise must be made by experts from that field and not only by anti-plagiarism programs. Since resulting similarity coefficients vary according to the anti-plagiarism software, it follows that our software can be a good indicator for the performance of software existing on the market to detect plagiarism. It also can be developed with a vocabulary that is richer in synonyms and with a more advantageous interface, as well as to avoid changes in citation. Likewise, it can be improved by creating a script section for keeping quotes in the document, since quotes are important and should not be modified. For the time being, the software can only be used for texts written in Romanian, but it can be easily adapted to any other language.

References

1. A.E. Shamoo, D.B. Resnik, *Responsible Conduct o Research*, Oxford University Press, (2009)
2. M. Potthast, B. Stein, A. Barrón-Cedeño, P. Rosso, An Evaluation Framework for Plagiarism Detection, *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING 2010 Beijing, China, (2010)
3. D. Pecorari, Good and original: Plagiarism and patchwriting in academic second-language writing, *Journal of Second Language Writing* 12, p.317–345, (2003)
4. J. Mariani, G. Francopoulo, P. Paroubek, A Study of Reuse and Plagiarism in Speech and Natural Language, *Processing papers, BIRNDL 2016* Joint Workshop on Bibliometric-enhanced Information Retrieval and NLP for Digital Libraries, (2016)
5. W. Sutherland-Smith, R. Carr, Turnitin.com: Teachers' Perspectives of Anti-Plagiarism Software in Raising Issues of Educational Integrity, *Journal of University Teaching and Learning Practice,* 2(3), (2005)
6. M. Vila Rigat, *Paraphrase Scope and Typology. A Data-Driven Approach from Computational Linguistics*, Phd Thesis, Universidad de Barcelona, 2013.
7. *** Dictionar online (Online Dictionary), available at: http://dexonline.net, accessed December 2016.
8. *** *PlagScan*, available at: http://www.plagscan.com/plagiarism-check/, accessed January 2017
9. *** *Plagiarism Checker*, available at: http://smallseotools.com/plagiarism-checker/, accessed January 2017
10. *** *Plagiarisma*, available at: http://plagiarisma.net, accessed January 2017.
11. *** *Sistem antiplagiat*, available at: http://sistemantiplagiat.ro/pagina-principala, accessed February 2017.