UDC: 68 1.5.004.42 DOI 10.36910/775.24153966.2022.74.4

L. Gumeniuk[1], V. Lotysh[1], Y. Vashkurak[2], P.Humeniuk[1]
[1]Lutsk National Technical University
[2]Lviv Polytechnic National University

AUTOMATION OF THE SEARCH OF UNCLEAR DUPLICATE TEXT ELECTRONIC DOCUMENTS

The process of searching for duplicate text documents in the Ukrainian language is automated. The existing approaches to the determination of duplicate text documents are analyzed. The software implementation of the main indicators of text similarity has been carried out. The shingle algorithm and its software implementation are presented. The algorithm for automating the search for fuzzy duplicates is implemented in software. The resulting software is tested on test cases

Keywords: *automation, software, algorithm, shingles, duplicates, comparison.*

Л.О. Гуменюк, В.В. Лотиш, Ю.З. Вашкурак, П.О Гуменюк

АВТОМАТИЗАЦІЯ ПОШУКУ НЕЧІТКИХ ДУБЛІКАТІВ ЕЛЕКТРОННИХ ТЕКСТОВИХ ДОКУМЕНТІВ

У роботі автоматизовано процес пошуку дублікатів текстових документів українською мовою. Проаналізовано існуючі підходи до визначення дублікатів текстових документів. Здійснена програмна реалізація основних покажчиків подібності текстів. Представлено алгоритм шинглу та його програмна реалізація. Програмно реалізовано алгоритм автоматизації пошуку нечітких дублікатів. Отримане програмне забезпечення перевірено на тестових прикладах.

Ключові слова: *автоматизація, програмне забезпечення, алгоритм, шингли, дублікати, порівняння.*

Statement of the problem. Nowadays there is an acute problem of duplication of information in the network. Similar objects can be web pages of the same subject (information search), customers with the same preferences (users of services for watching movies, listening to music with the same preferences in movies or music), duplicate texts or other content (plagiarism detection, spam filtering). Search engines and website owners suffer the most from the problem of duplicate information, as the former are forced to constantly index information on the web to keep it up-to-date, and the presence of duplicates slows down the construction of the index and searching through it, thereby reducing the desire to use this search engine; and the latter suffer from duplicate information that clogs their databases, which forces them to filter the data in a certain way so that users do not see the same information.

A document that is a slightly modified copy of the original is called a fuzzy duplicate. Detecting fuzzy duplicate documents is a challenging task. Search engines, due to the huge amount of data stored in them, cannot solve this task by simply comparing all the texts of documents. Therefore, they are forced to reduce the cost of duplicate detection by applying various methods of approximate document representation, which may lead to deterioration of the quality of duplicate detection. In web search tasks, the most accurate separation of web pages from their content is also an important factor.

Thus, the automation of methods for determining fuzzy duplicate texts resistant to changes in documents is an urgent task.

Analysis of the latest research and publications. Today there are a large number of methods and algorithms for detecting borrowings based on syntactic and lexical principles of document construction [1-3].

In 1997, a new approach to duplicate search was developed, in which each document is represented as a set of "shingles", which are a certain number of consecutive words [4-6]. The similarity of documents was determined based on the intersection of the set of their shingles. The advantage of this approach is that short and long documents are the same number of values. This allows to simply compare the values of different documents in pairs.

Another approach to duplicate detection is based on lexicon [7-9]. This method has high computational efficiency, which is superior to the algorithm [4], and shows good performance in relatively small documents.

There is also a signature approach to finding duplicates [10]. It starts with the selection of a set of words called the "descriptive set". For each word, a frequency threshold is fixed and for each document a vector is calculated, the i-th component of which is equal to 1 if the relative frequency of the i-th word from the "descriptive set" is greater than the selected frequency threshold, 0 - if less. This binary vector is the document signature. If documents have the same signature, they are considered similar.

Setting of tasks. Fuzzy duplicate search algorithms are widely used in optimizing the work of information retrieval systems and for automatic classification / clustering of documents. Automation of this process aims to significantly speed up the search for duplicates in document arrays, improve the quality and accuracy of such search.

Presentation of the main material. To work effectively with text data and search for duplicates, it is necessary to apply methods of cleaning and pre-processing of texts to bring the data to the same form, to highlight the main thing. To clean the texts, the removal of punctuation marks and stop words that do not carry significant information about the texts is used [11]. Stemming [12] or lemmatization [13] is used to bring texts to a single form. Stemming is a procedure for determining the base of a given word, which may not coincide with its morphological root. Lemmatization is the process of bringing the word to its form that is contained in the dictionary.

In this paper, stemming is used for the Ukrainian language.

Searching for similar items based on fixed numerical criteria using a query language is a common operation in databases, search engines and many other applications. But when you need to formally compare words or sentences, other approaches are used: editorial distance or vector similarity.

Editorial distances can help when you need to formally compare words or sentences and understand which ones are more similar to each other. The main concept in calculating editorial distances is an operation. There are only four operations: delete, insert, replace, rearrange. Each operation can involve one character in a string. To find out the editorial distance between two strings, you need to calculate the minimum number of operations that need to be done to turn the first string into the second.

There are several basic editorial distances, the main difference between which is the set of operations that can be used. The most commonly used are the Hamming, Levenshtein, Damerau-Levenshtein, and Jaro-Winkler editorial distances.

The vector similarity method allows to translate the similarity between objects as they are perceived into vector space. This means that when we represent images or text fragments as vector embeddings, their semantic similarity is expressed by how close their vectors are in space. The user needs to calculate the distance between these vectors in vector space according to the distance search method that best suits the task.

The main methods of distance search by vector method include Q-gram distance; Jaccard distance; cosine distance; Sorensen distance.

For the software implementation of the described methods of searching for similar elements of text data, the open, free pytextdist library of the Python programming language was used. The library includes procedures for calculating various text distance and similarity measures. The functions in this package take two strings as input and return the distance/similarity metric between them.

The following functions are used:

for editorial distance – levenshtein_distance, levenshtein_similarity; lcs_distance, lcs_similarity; damerau_levenshtein_distance, damerau_levenshtein_similarity; hamming_distance, hamming_similarity; jaro_similarity, jaro_winkler_similarity;

for vector similarity – cosine_similarity; jaccard_similarity; sorensen_dice_similarity; qgram_similarity.

We also used the standard module (included in Python) for determining the similarity measure difflib and the function for obtaining comparisons using the shingle method genshingle.

The software package for analyzing the main similarity indicators consists of three scripts:

1_write.py – script for wr.csv file with titles of literary works. The script takes 500 records (author, title, edition, etc.) and generates a file with only the titles of works.

2_DataFrame.py is used to obtain a data frame from the found similarity pointers. The script receives lines from the results file (1_write.py) and in a loop forms a DataFrame with the results of calculating the similarity pointers. For this purpose, the library elements pytextdist.vector_similarity and pytextdist.edit_distanc are used. The Pandas package is used to create the DataFrame. The data obtained as a result of the calculation are stored in the file date_file.pkl.

For morphological analysis in Python, we used the pymorphy2 morphological analyzer [15]. Using the pymorphy2 property, a function was written to search for text duplicates, words and sentences are recognized as duplicates if they match by 50 percent or more.

Based on the above, the shingle method and the diff class were chosen for further detailed consideration.

A shingle is a small fragment of text consisting of several words, processed using a special technique for analysis. These are separate parts (substrings), selected for comparison from the main part of the text, with a certain number of words in their sequence.

For effective comparison, it is necessary to set the correct algorithm parameters. Small shingles require more time to process, but give high accuracy. The smallest consists of 3 words, the largest - of 8. Shingles of more than 8 words are not suitable for determining uniqueness because they are inefficient.

Next, the checksum of the text is calculated. Checksums can be obtained by hashing using different algorithms (SHA1, SHA3, CRC32, MD5). Next, it is necessary to evaluate the coincidence of the obtained checksums for the two compared texts.

The principle of the shingle algorithm is based on comparing randomly selected shingle checksums (subsequences) of two documents.

Comparing each of the elements of both documents reveals the ratio of the same values, which makes it possible to determine the level of identity, or uniqueness, of each of the texts.

The uniqueness of the text is calculated by the percentage of unique shingles. For example, if the text consists of 100 shingles and 95 of them are unique, then the uniqueness of the text is 95%.

In the software implementation of the shingles algorithm in Python, the binascii library is used to calculate checksums. It includes the binascii.crc32(data[, value]) function.

To test the effect of the number of words in the shingle on the search accuracy, two scripts were created. The first script shingle.py loads data from the file wr.csv, where there are 250 000 records. After loading, the similarity of records is determined and the time spent on checking is recorded. The second script shingle_analiz.py works with files of pkl format, which stores the results of the previous script. After loading the data, visualization is performed using the matplotlib library.

The result of the calculations is presented in the form of a graph (dependence of the search time on the number of words in the shingles) - Figure 1.

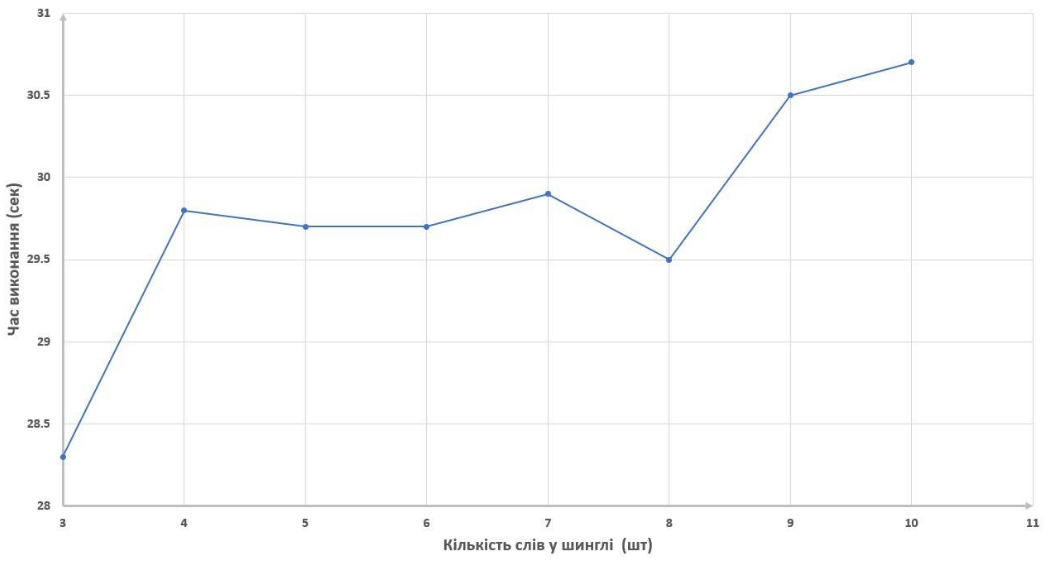


Fig. 1. **Dependence of execution time on the number of words in shingles**

The analysis of the results showed that the speed of the shingle algorithm for determining the editorial distance does not significantly depend on the number of words in the shingle (for texts that include 250,000 units of information). The accuracy of the editorial distance determination improves with the shingle length and becomes optimal at the shingle length equal to 8 words.

Automation of the process of searching for fuzzy duplicates in electronic text documents consists of a number of stages:

- converting electronic text documents of docx and pdf formats into txt text documents;
- search for fuzzy duplicates by the shingle method;
- search for fuzzy duplicates using the difflib module.

Convert electronic text documents of docx, pdf format to txt format.

We form a list of docx and pdf files for transcoding. It is necessary to specify the directory where the files are located. To do this, create an interactive dialog using the tkinter module, and in it the filedialog function. Having received the path to the directory with the source files, we form separate lists of docx and pdf files. The file_docx list is for docx files and file_pdf for pdf files, respectively. The resulting lists will be used to convert files to txt format. The list includes the full path to the file and the file name.

To convert docx documents we use python-docx [16].

To transcode pdf files we use the pdfplumber library [17].

The path to the newly created files (txt format) is written to the text file spysok.txt. It contains information about the path and name of the txt file.

Search for fuzzy duplicates using the shingle method.

This method, algorithm and software implementation are described above. Let us use the obtained results to create a script to detect fuzzy duplicates among many documents.

The list of documents for comparison is in the file spysok.txt.

The check algorithm is as follows - read the list of files from spysok.txt into a variable of type list. Find the number of records in the variable. Set the name and location of the file that should be compared with the files from the list. In the loop we load the files from the list and in the shingle procedure we determine the similarity value of the given file with the files from the list. The result is written to a pdf file, for which we use the reportlab library.

In the main procedure for implementing this algorithm in the for i in range(0,k) loop, the similarity value is checked. The file_name variable takes the value of the file that is compared with the rest of the files from the data list. The comparison is performed using the genshingle and compaire procedures.

To create a report in PDF format, the following components of the ReportLab library are connected:

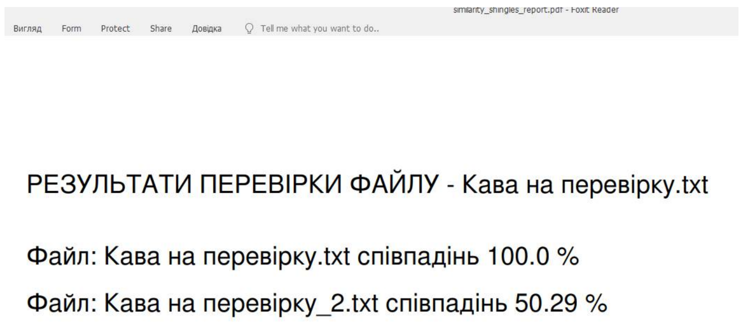
```
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.pagesizes import letter
from reportlab.lib.units import inch, mm
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle as PS
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
```

To work with Ukrainian-language texts, the text was canonized (canonize procedure) before comparison.

To test the resulting script, we took two files "Coffee to check.txt" and "Coffee to check_2.txt" obtained from the corresponding docx files. In the file "Coffee to check_2.txt" half of the text is completely replaced.

Thus, comparing the file "Coffee to check.txt" with itself and with the file "Coffee to check_2.txt" we should get similarities of 100% and 50% respectively. Here are the results of the test.

Figure 2 shows a screenshot of the file similarity_shingles_report.pdf, which is a report:


similarity_shingles_report.pdf - Foxit Reader
Вигляд
Form
Protect
Share
Довідка
Tell me what you want to do..

РЕЗУЛЬТАТИ ПЕРЕВІРКИ ФАЙЛУ - Кава на перевірку.txt

Файл: Кава на перевірку.txt співпадінь 100.0 %

Файл: Кава на перевірку_2.txt співпадінь 50.29 %

Fig 2. **Results of the test verification of files.**

As you can see from the results, the script for searching for duplicates using the shingle method coped with the test task perfectly.

Search for fuzzy duplicates using the difflib module.

The module provides classes and functions for comparing sequences. It can be used, for example, to compare files and provide information about file differences in different formats, including HTML, context, and unified differences.

In our case, we will use the SequenceMatcher class for comparison. This is a flexible class for comparing pairs of sequences of any type, provided that the sequence elements are hashed. The basic algorithm precedes the algorithm published in the late 1980s by Ratcliffe [18] and Obershelp called "Gestalt pattern matching" [19].

The SequenceMatcher class supports a heuristic that automatically considers certain elements of a sequence as undesirable. The heuristic counts how many times each individual element occurs in the sequence. If duplicates of an element (after the first one) account for more than 1% of the sequence, and the sequence length is at least 200 elements, this element is marked as "popular" and treated as undesirable for sequence matching purposes. This heuristic can be disabled by setting the autojunk argument to False when creating the SequenceMatcher [20].

In general, this algorithm is similar to the search for fuzzy duplicates using shingles, but in addition to the use of the SequenceMatcher class, it is supplemented by a system of preliminary processing of files. After the canonization procedure, stemming is applied to the text file. In our case, we use the UkrainianStemmer class [21].

In parallel, a "bag of words" is formed. In this approach, the text (document) is represented as a "bag" of its words, without taking into account the grammar and even the order of words, but with preservation of multiplicity. In fact, it describes the presence of words in the text. The "bag" shows which words are found in the text, not where they are.

After canonization and stemming, the created "bags" of words are compared in the similarity procedure. This procedure is used to check SequenceMatcher. This process is repeated until all files with which the base file is compared are exhausted.

Then a report on the presence of matches is generated.

The resulting script was checked using the same method and with the same data as for the shingle method. With a small number of shingles (3), the results are close (50.29% - shingle method; 53.8% - difflib module).

Conclusions. Thus, the whole process of searching for fuzzy duplicates in electronic text documents is automated: converting electronic text documents in docx and pdf formats into text documents in txt format and creating a list of files for verification; searching for fuzzy duplicates using the shingle method (after receiving the results, the names of files whose uniqueness is <20% are removed from the initial list); searching for fuzzy duplicates using the difflib module with the formation of a report on the uniqueness of a given text file. The automation algorithm and software implementation of the search for fuzzy duplicates of text documents in Ukrainian are presented. The obtained results are analyzed.

List of references.

1. Manber U. Finding Similar Files in a Large File System // Proc. of the Winter USENIX Technical Conference, 1994. P. 1-10.

2. Heintze N. Scalable document fingerprinting // Proc. of the 2nd USENIX Workshop on Electronic Commerce, 1996. P. 191-200.

3. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. СПб.: Невский диалект, 2003. 656 с.

4. Broder A., Glassman S., Manasse M. and Zweig G. Syntactic clustering of the Web // Proc. of the 6th International World Wide Web Conference, 1997. P. 1157-1166.

5. Fetterly D., Manasse M., Najork M. A Large-Scale Study of the Evolution of Web Pages // Proc. of the 12th international conference on World Wide Web, 2003. P. 669-678.

6. Broder A., Charikar M., Frieze A., Mitzenmacher M. Min-wise independent permutations // Proc. of the thirtieth annual ACM symposium on Theory of computing, 1998. P. 327-336.

7. Chowdhury A., Frieder O., Grossman D., McCabe M. Collection statistics for fast duplicate document detection. // ACM Transactions on Information Systems (TOIS), 2002. Vol. 20, No. 2. P. 171-191.

8. Kolcz A., Chowdhury A., Alspector J. Improved Robustness of Signature-Based Near-Replica Detection via Lexicon Randomization // Proc. of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, 2004. P. 605-610.

9. Pugh W. Detecting duplicate and near - duplicate files [Електронний ресурс]. URL: http://www.cs.umd.edu/~pugh/google/Duplicates.pdf (дата звернення: 15.07.22).

10. Ilyinsky S., Kuzmin M., Melkov A., Segalovich I. An efficient method to detect duplicates of Web documents with the use of inverted index // Proc. of the 11th International World Wide Web Conference, 2002.

11.Стоп-слова [Електронний ресурс]. URL: https://alexsmokinof.lviv.ua/%D1%81%D1%82%D0%BE%D0%BF-%D1%81%D0%BB%D0%BE%D0%B2%D0%B0/ (дата звернення: 15.07.22)..

12.Dawson J. Suffix removal for word conflation // Bulletin of the Association for Literary and Linguistic Computing, 1974. Vol. 2 No. 3. P. 33-46.

13.Глибовець А. М. Алгоритм токенізації та стемінгу для текстів українською мовою / А. М. Глибовець, В. В. Точицький. // НАУКОВІ ЗАПИСКИ НаУКМА. Комп'ютерні науки. – 2017. – №198. – С. 4–8.

14. Korobov M.: Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts, pp 320-332 (2015).

15. Морфологический анализатор pymorphy2 [Електронний ресурс]. URL: https://pymorphy2.readthedocs.io/en/stable/ (дата звернення: 15.07.22).

16. Project description [Електронний ресурс]. URL: https://pypi.org/project/python-docx/ (дата звернення: 15.07.22).

17. LIB4DEV.IN [Електронний ресурс]. URL: http://www.lib4dev.in/info/jsvine/pdfplumber/41279279 (дата звернення: 15.07.22).

18. John W. Ratcliff and David Metzener: Pattern Matching: The Gestalt Approach, Dr. Dobb's Journal, Issue 46, July 1988

19. Paul E. Black, "Ratcliff/Obershelp pattern recognition", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 8 January 2021. (accessed TODAY) Available from: https://www.nist.gov/dads/HTML/ratcliffObershelp.html

20. difflib — Helpers for computing deltas inside the Python documentation [Електронний ресурс]. URL: https://docs.python.org/3/library/difflib.html (дата звернення: 15.07.22).

21. Stemmer for Ukrainian language in Python [Електронний ресурс]. URL: https://github.com/Amice13/ukr_stemmer (дата звернення: 15.07.22).