Olena Kuropiatnyk[1], Dmytro Lehkokonets[2]
Ukraiński Państwowy Uniwersytet Nauki i Technologii, Dniepr

https://doi.org/10.29316/9788368103281_2

EXPERIMENTAL INVESTIGATION OF THE PROGRAM CLONE DETECTION ALGORITHMS EFFECTIVENESS

EKSPERYMENTALNE BADANIE SKUTECZNOŚCI ALGORYTMÓW WYKRYWANIA KLONÓW PROGRAMÓW

Abstract

The paper reviews existing algorithms and methods for detecting program clones. The authors conducted a series of experiments on detecting clones in the texts of educational project programs written in C++. For this purpose, a software application was created - a program clone analyzer. It implements the Jaccard Similarity, Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) Similarity and Winnow (Winnowing) Similarity algorithms and provides the ability to perform 1:1 and N:M program text comparisons. During the experiment, the application was used to measure the running time of the algorithms and evaluate the time efficiency based on SR-estimates. The results obtained allow for a reasonable choice of clone detection algorithms taking into account their time efficiency in a given software and hardware environment.

Keywords: algorithms effectiveness, program clone, information technology, software, software quality

[1] https://orcid.org/0000-0003-2286-884X, e-mail: olena.kuropiatnyk@ust.edu.ua
[2] https://orcid.org/0009-0004-1411-3564, e-mail: sulzlzossd@gmail.com

Streszczenie

W rozdziale dokonano przeglądu istniejących algorytmów i metod wykrywania klonów programowych. Autorzy przeprowadzili szereg eksperymentów nad wykrywaniem klonów w tekstach programów edukacyjnych napisanych w języku C++. W tym celu stworzono aplikację – analizator klonów programowych. Implementuje ona algorytmy podobieństwa Jaccarda, podobieństwa Running-Karp-Rabin Greedy-String-Tiling (RKR--GST) oraz podobieństwa Winnow (Winnowing) i umożliwia przeprowadzanie porównań tekstów programów w proporcjach 1:1 i N:M. W trakcie eksperymentu aplikacja została wykorzystana do pomiaru czasu działania algorytmów i oceny efektywności czasowej na podstawie estymacji SR. Uzyskane wyniki pozwalają na rozsądny wybór algorytmów wykrywania klonów, biorąc pod uwagę ich wydajność czasową w danym środowisku programowym i sprzętowym.

Słowa kluczowe: skuteczność algorytmów, klon programu, technologia informacyjna, oprogramowanie, jakość oprogramowania

Introduction

Program (code) clones are duplicates or similar fragments of code within a software project (Martinez-Gil, 2024). Similarity in fragments can manifest itself at different levels: semantic (identical or similar functionality) and syntactic (similarity in the text of programs by type and sequence of tokens).

The presence of clones can have a negative impact on maintain processes, as well as cause a number of problems directly during software development and operation. Clones increase the volume of software code and the time for its testing, complicate program debugging (errors must be localized and corrected in each clone separately). If their correct replacement with software entities (functions, modules) with reuse (call) without duplication does not change the functionality of the program, then such cases should be subject to refactoring. Clones can carry additional danger if they are vulnerable to attacks. The threat increases in proportion to the number of duplicate fragments that are vulnerable. This is the reason for the work to eliminate clones (Zhang, Sakurai, 2021). The presence of clones may indicate a violation of the principles of academic integrity when they are detected in the academic works of various students. In this case, the task of detecting clones can be considered as a separate task for detecting plagiarism. The problem of clones

increases with the growth of the volume of program code and the increase in branches in collective development, therefore, it is advisable to use automated means of their detection. Currently, there are a significant number of algorithms and methods for detecting clones (Shobha, Rana, Kansal Tanwar, 2020; Wang, Ye, Zhang, Xue, Liu, 2023). It is advisable to have selection criteria and substantiated evaluations of methods and algorithms which obtained based on them for use in different contexts.

This work is devoted to the experimental assessment of the effectiveness of algorithms for detecting program clones. The object of this research is the processes of detecting program clones. The research subject is algorithms for detecting program clones. The purpose of the work is to analyze the time efficiency of various known algorithms for detecting clones for their further justified choice in various tasks. To achieve this goal, the authors have developed special software – analyzer application.

Related works

Program clones can have different natures. In this regard, clones are divided into the following types (Walker, Cerny, Song, 2020; Shynkarenko, Kuropiatnyk, Demidovich, 2025; Jo, Lee, Yoo, 2021):

1) syntactically identical code fragments, except for non-printing symbols and comments,
2) syntactically identical code fragments and, in addition to the differences of type 1 clones, differences in identifiers and literal values are allowed,
3) syntactically similar code fragments that differ at the expression level and have restrictions of types 1 and 2. Expressions can be added to these fragments, changed and/or deleted relative to each other,
4) semantically similar, but syntactically dissimilar code fragments. These include different algorithms that solve the same problems. For example, sorting or linear and binary search algorithms,
5) syntactically similar fragments that have different functions. This type was discovered in (Shynkarenko et al., 2025) and called the inverted fourth type.

A common classification of clone detection methods is based on the approach to the presentation and transformation of the program text for similarity analysis. Here, the following categories of methods are distinguished: textual, token, syntactic (tree-based, metric-based), semantic, and machine learning-based (Walker et al., 2020; Pravorska, Barmac, Medzatiy, 2021). Machine learning-based methods require appropriate training bases, which further complicate their work.

Each of the above categories provides a number of methods suitable for detecting clones of a certain type (Shynkarenko et al., 2025). Most methods (except for text and some syntactic methods based on metrics) involve building a new representation of the program, which will then be used to compare texts to detect clones. Among them, the most common, but at the same time laborious and difficult to scale in the future, are representations based on graphs and trees, namely: Abstract Syntax Trees (ASTs) Similarity, Code Embeddings Similarity (based on ASTs and collections of paths in trees (Alon, Zilberstein, Levy, Yahav, 2018), Graph-based Similarity, Program Dependence Graph Similarity (Martinez-Gil, 2024). There are a number of similarity detecting methods (including methods of clones similarity detecting) (Martinez-Gil, 2024) that do not create program entities that directly reproduce the structure when processing the input text. There are Bag-of-Words Similarity, Jaccard Similarity, Levenshtein Similarity, N-grams Similarity, Rolling Hash Similarity, Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) Similarity, TF-IDF Similarity, Winnow Similarity, and others.

Some of them are used to detect duplicates in both natural language texts and program texts, in particular for detecting plagiarism. They are the basis of software services. For example, Moss detects structural similarities between programs (Aiken, 2022). Moss uses a technique known as document fingerprinting (Zamrii, Kulakov, 2024), where a set of hashes (so-called "fingerprints") is pre-calculated for each document, and only then comparison is made between these fingerprints, which dramatically reduces the amount of calculations. In practical implementation, MOSS uses an improved version of this technique – the Winnowing (Winnow Similarity) algorithm.

The source code plagiarism detector JPlag is one of the popular source code plagiarism detection systems (Wise, 1993). JPlag compares the structure

of programs by extracting a layer of abstraction from their parse trees. Thus, it is resistant to attempts to hide plagiarism. JPlag has open source code and can be executed locally, thus complying with GDPR.

Existing methods for detecting software clones are studied in theoretical and practical planes (Martinez-Gil, 2024; Zamrii, Kulakov, 2024; Ain et al., 2019). The first involves studying the general principles, the features of the necessary program text transformations, analyzing the complexity of the corresponding algorithms based on the generalization of already known results of research conducted by other scientists, etc. The second is experimental measurements of various performance indicators of the methods, the main of which are accuracy and the time required to measure similarity. On their basis, conclusions are drawn regarding the use of methods for detecting different types of clones.

In this paper, experimental investigations of the effectiveness of clone detection methods with high accuracy and low computational time (Martinez-Gil, 2024) and detecting clones of types 1-3 (Ain et al., 2019), namely Jaccard Similarity, Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) Similarity and Winnow (Winnowing) Similarity. Investigations were conducted with the aim of making a reasonable choice of algorithms based on SR estimates, as well as identifying the influence of the software and hardware environment on their operation.

Methods

Program clone detection algorithms

Jaccard Similarity is used to calculate the similarity between two objects, in particular two text documents. This method can be used to find the similarity between two asymmetric binary vectors or to find the similarity between two sets (Fletcher, Zahidul Islam, 2018).

Jaccard Similarity can be calculated as the size of the intersection divided by the size of the union of the two sets:

$$J(A,B)=\frac{|A \quad B|}{|A \quad B|}, \tag{1}$$

where $|A \quad B|$ is number of common elements between sets A i B, $|A \quad B|$ is number of elements in the union of sets A i B.

The Jaccard similarity calculation algorithm has wide application. When working with texts (in particular, texts of programs for detecting clones) it can be reduced to the following:

1) break the text into tokens (words, symbols or code fragments),
2) calculate the intersection of tokens between two texts as the number of tokens found in both texts,
3) calculate the union of tokens – the number of unique tokens in both texts,
4) calculate the similarity using (1).

RKR-GST (Running Karp-Rabin Greedy String Tiling) is an algorithm for searching for repeated substrings in a text that uses heuristics to efficiently find maximum matches.

RKR-GST combines two approaches (Fletcher, Zahidul Islam, 2018):

1) Greedy String Tiling (GST) is a greedy algorithm that finds the longest matches between two non-overlapping strings (or documents),
2) Running Karp-Rabin Matching is an optimized version of matching using Karp-Rabin hashing, which allows you to quickly search for substrings by hashes.

Different measures (coefficients) can be used to estimate similarity in RKR-GST.

In this work, the average similarity measure calculated using the Dice coefficient was used to estimate the similarity of program texts by the RKR-GST algorithm:

$$\mathrm{sim}(A,B) = \frac{2 \cdot coverage}{\mathrm{size}(A) + \mathrm{size}(B)}, \qquad (2)$$

where coverage is total length of all matches found, size( ) is length of one of the compared strings.

Formula (2) is a standard for calculating similarity between sets of tokens and takes into account all their differences. It is symmetric: regardless of which document is larger, the result is the same for comparing A with B and

B with A. This approach provides a balanced consideration of information from both files and is well suited for general similarity assessments, where the results should take into account both the additional unique material in one of the files and their common part. This measure is best used when all differences are equally important.

Winnowing is an algorithm used to create a document fingerprint. The document contains an input string, either the original source code or tokenized code. The algorithm preprocesses the input string, removing all whitespace. After that, n-grams are formed from the resulting string, which are hashed to create a sequence of hashes. This set of hashes is divided into a series of overlapping windows of a certain size. For each window, the hash with the lowest value is selected as its representative. This significantly reduces the number of hash values and allows for more efficient detection of similar substrings. Also, hash sets are precomputed for each input string separately and stored in memory instead of being recalculated for each pairwise comparison. This speeds up the similarity detection and measurement phase (Schleimer, Wilkerson, Aiken, 2003).

The formulas for polynomial and rolling hashing (Shrestha, Gautam, Sharma, Bhandari, 2023) are used to calculate the hash. Rolling hashing allows for efficient hash update when moving to the next substring, using a previously computed hash (Faisal et al., 2020).

Determination of time efficiency indicators for the implementation of clone detection algorithms

The investigation of the effectiveness of clone detection algorithms was carried out on the basis of determining the time efficiency indicators (Shynkarenko, 2009):

- the average advantage of one algorithm over another (S-score),
- the ratio of the areas of advantage of each of the algorithms (R-scores).

The determination of time efficiency indicators is carried out on the basis of a computer experiment. The experiment consists of a sequence of algorithm tests. Each test consists of processing program texts and performing the

operation of calculating the similarity of program texts by the corresponding algorithm with time measurement.

The calculation of the S-score is performed as follows

$$S = \frac{1}{N} \sum_{i=1}^{N} \frac{t_i^{''} - t_i^{'}}{max\left(t_i^{'}, t_i^{''}\right)} \cdot 100\%, \tag{3}$$

were $t_i^{'}, t_i^{''}$ are processing operation execution time for each of the algorithms in the i-th trial, N is number of trials.

If the obtained S-score is positive, this is an advantage of the first algorithm, if the score is negative, this is an advantage of the second.

The R-score is calculated as follows

$$R = \frac{1}{N} \sum_{i=1}^{N} sign\left(t_i^{''} - t_i^{'}\right) \cdot 100\%, \; sign(a) = \begin{cases} 1, if \;\; a > 0 \\ 0, \; if \;\; a \le 0 \end{cases}, \tag{4}$$

If the R-score is more than 50%, then the first algorithm has an advantage, if less – than the second. If the R-score is equal to 50%, the areas of advantage are equal.

Performing a series of trials increases the statistical accuracy of the result.

To measure the operation execution time, a high-precision system timer is used – QueryPerformanceTimer (C++ language). Using the timer involves:

- obtaining the frequency of the timer frequency,
- obtaining the starting value of the timer time_start,
- obtaining the final value of the timer time_finish,
- calculating the time of running algorithms.

Results

Software application development

To investigate the time efficiency of clone detection algorithms, a software application was developed in C++. The main functionality of the application is specified using a use case diagram, which is shown in Figure 1. The

application has functionality for both conducting trial series and calculating SR scores, as well as searching for clones in 1:1 files.

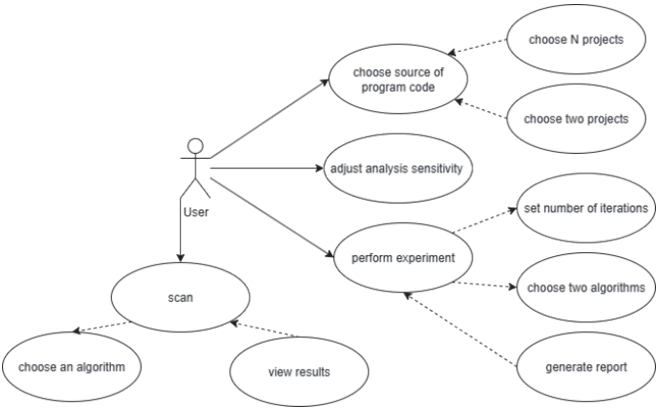


Figure 1. Use-case diagram
Source: own study.

The application is built and implemented using the principles of object-oriented design and programming. The application design, which is created according to UML principles, is shown in Figure 2. The above presentation omits the detailed class structure (attributes and methods) to simplify understanding of the basic structure of the application.

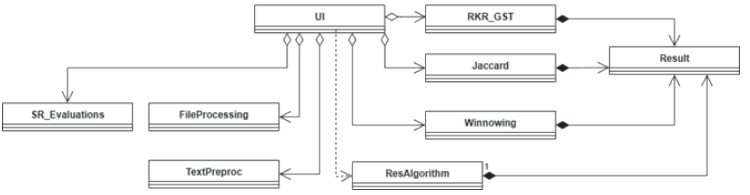


Figure 2. Object-oriented application model
Source: own study.

The classes have the following responsibilities:

- UI – implementation of user interaction with the program,
- RKR_GST, Jaccard, Winnowing – implementation of clone detection algorithms described in the previous section,
- Result – storage of results of comparison of a pair of program texts,

- SR_Evaluation – calculation and storage of SR-scores,
- FileProcessing – Reading and processing of C++ projects selected by the user,
- TextPreproc – preprocessing of the program text according to selected parameters, namely: ignoring absences, variable names, comments, case,
- ResAlgorithms – storage of results of a series of trials (scans) using a specific algorithm.

The application has a graphical user interface. A form is chosen as the structure of the dialog. A computer "mouse" is enough to interact with the form.

Experimental investigation of the effectiveness of algorithms

Experiment 1

The purpose of the experiment is to determine the influence of the software and hardware environment on the time efficiency of the Jaccard and RKR-GST algorithms and to compare their indicators.

The experimental base consists of educational projects in the discipline "General-purpose software" for the specialties Computer Engineering and Cyber Security of the Bachelor's degree of the educational and scientific institute "Dnipro Institute of Infrastructure and Transport" of the of the Ukrainian State University of Science and Technology. The projects consist of program texts in the C++ language in the amount of:

- projects of the first type – 26 projects, the volume of one file, up to 150 lines of code,
- projects of the second type – 27 projects, the volume of 1-5 files, up to 200 lines of code.
- The following requirements were imposed on the programs of the first type of projects:
    - a program for processing a matrix, which is presented in the form of struct in C++,
- each fundamentally new action should be presented in the form of a function. The following set of functions is mandatory:

  - main – the main function of the program,
  - create – creating a matrix (specifying the dimension, allocating memory),
  - delete – clearing the memory occupied by the matrix,
  - processing – processing the matrix, which should not contain input and/or output operators,
  - manualFilling, randomFilling – filling the matrix from the keyboard and using a random number generator, respectively,
  - show – displaying the matrix on the screen,
- provide for the possibility of entering the dimension of the matrix, choosing the method of filling it from the keyboard and using a random number generator,
- comment on the matrix processing function(s), based on the logical subtasks into which the main task is divided,
- it is forbidden to use auxiliary arrays and matrices, including STL-containers.

The second type of projects involved a program that included the implementation of classes and functions according to the individual task variant, as well as the implementation of the main function of the program to demonstrate working with class objects and friendly entities (classes/functions). Each class must have:

- at least three attributes,
- a default constructor,
- a constructor with parameters,
- a copy constructor,
- a destructor,
- setters, getters,
- a toString method that returns a string containing the names and values of all attributes.

Table 1 provides a description of the hardware on which the experiment was conducted.

The experimental methodology involves the following steps:

1) select the number of projects N, algorithms 1 and 2 for clone detection,

2) compare the i-th project with all using algorithm 1, save the algorithm running time for each comparison,
3) based on the data in p. 2, find the arithmetic mean value of the algorithm running time and save it in the i-th element of the array of average values for the algorithm under study,
4) compare each subsequent (i+1) project as described in p. 2-3,
5) repeat p. 2 – p. 4 for the second algorithm,
6) in the arrays obtained in p. 3, series of numbers are saved, which are the average values of the execution time of algorithms 1 and 2. Based on the values of the elements of these arrays, calculate SR-estimates according to (3), (4).

Table 1. Hardware

Component	Characteristics		
	Comp 1	**Comp 2**	**Comp 3**
CPU	12th Gen Intel(R) Core(TM) i5-1235U (1.30 GHz) Cores: 2P + 8E Cache: L1 Data 2*48KB+8*32KB L1 Inst. 2*32KB + 8*64KB L2 2*1.25MB + 2*2MB L3 12MB	Intel(R) Core(TM) i5-10400F (2.90 GHz) Cores: 6 Cache: L1 Data 6*32KB L1 Inst. 6*32KB L2 6*256KB L3 12MB	Intel(R) Core(TM) i5-9400F (2.90 GHz) Cores: 6 Cache: L1 Data 6*32KB L1 Inst. 6*32KB L2 6*256KB L3 9MB
RAM	DDR4 16 GB	DDR4 16 GB	DDR4 16 GB
Type storage device	SSD	SSD	SSD
GPU	Intel(R) Iris(R) Xe Graphics (128 MB)	NVIDIA GeForce GTX 1050Ti (4GB)	NVIDIA GeForce GTX 1650 (4GB)
Operating system	Windows 10 Home x64	Windows 11 Home x64	Windows 10 Home x64

Source: own study.

The experimental results were obtained with the following input parameters:

- algorithm 1 – „Jaccard”,

- algorithm 2 – “RKR-GST” with the minimum match length parameter 3.

The obtained estimates of the time efficiency of the algorithms:

- comp 1: S≈57,48%, R≈100%,
- comp 2: S≈52,86%, R≈100%,
- comp 3: S≈53,37%, R≈100%.

Experiment 2

The purpose of the experiment is to determine the influence of the software and hardware environment on the time efficiency of the Jaccard and Winnowing algorithms and to compare their indicators.

The experimental base coincides with the base of experiment 1.

The experimental methodology coincides with the methodology of experiment 1.

The results of the experiment with the value of the input parameters:

- algorithm 1 – „Jaccard”,
- algorithm 2 – “Winnowing” with the K-gram length parameter of 3, the minimum match length parameter of 3.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈6,11%, R≈88,68%,
- comp 2: S≈7,7%, R≈92,45%,
- comp 3: S≈10,28%, R≈100%.

Experiment 3

The purpose of the experiment is to determine the influence of the software and hardware environment on the time efficiency of the RKR-GST and Winnowing algorithms and to compare their indicators.

The experimental base coincides with the base of experiment 1.

The experimental methodology coincides with the methodology of experiment 1.

The results of the experiment with the value of the input parameters:

- algorithm 1 – “RKR-GST” with the minimum match length parameter 3,

- algorithm 2 – “Winnowing” with the K-gram length parameter 3, the minimum match length parameter 3.

The obtained estimates of the time efficiency of the algorithms:

- comp 1: S≈-55,35%, R≈0%,
- comp 2: S≈-47,94%, R≈0%,
- comp 3: S≈-48,07%, R≈0%.

The following are the results of similar experiments for preprocessed data.

Experiment 4

This experiment is identical to experiment 1, except that preprocessing is applied to the text base, which allows ignoring indentations and comments. Clones detected by this approach can be attributed to the first type.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈55,06%, R≈100%,
- comp 2: S≈53,28%, R≈100%,
- comp 3: S≈52,52%, R≈100%.

Experiment 5

This experiment is identical to experiment 2, except that preprocessing is applied to the text base, which allows ignoring indentations and comments. Clones detected by this approach can be attributed to the first type.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈27,37%, R≈100%;
- comp 2: S≈29,32%, R≈100%;
- comp 3: S≈30,21%, R≈100%.

Experiment 6

This experiment is identical to experiment 3, except that preprocessing is applied to the text base, which allows ignoring indentations and comments. Clones detected by this approach can be attributed to the first type.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈-42,85%, R≈0%,
- comp 2: S≈-41.59%, R≈0%,

- comp 3: S≈-35,56%, R≈1,89%.

Experiment 7

This experiment is identical to experiment 1, except for the application of pre-processing to the text database, which allows ignoring indentations and comments, character case, variable names. Clones detected by this approach can be attributed to the second type.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈58,85%, R≈100%,
- comp 2: S≈54,51%, R≈100%,
- comp 3: S≈54,71%, R≈100%.

Experiment 8

This experiment is identical to experiment 2, except for the application of pre-processing to the text database, which allows ignoring indentations and comments, character case, variable names. Clones detected by this approach can be attributed to the second type.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈23,76%, R≈100%,
- comp 2: S≈27,91%, R≈100%,
- comp 3: S≈29,49%, R≈100%.

Experiment 9

This experiment is identical to experiment 3, except for the application of pre-processing to the text database, which allows ignoring indentations and comments, character case, variable names. Clones detected by this approach can be attributed to the second type.

The obtained estimates of the time efficiency of the algorithms*:*

- comp 1: S≈-42,2%, R≈0%,
- comp 2: S≈-36,97%, R≈0%,
- comp 3: S≈-37,24%, R≈0%.

Figures 3-5 show the visualization of the results by the C indicator. In the legend to the diagrams, ignore 1 indicates the case with preprocessing, which allowed the algorithms to ignore indents and comments, ignore 2 – in

addition to ignore 1, the case of characters and variable names. The bars of the chart show the S-score values. The R-score in the matched cases indicates the superiority of the same algorithm in each case. The same colors of the bars correspond to the same text bases with the same type of their preprocessing.

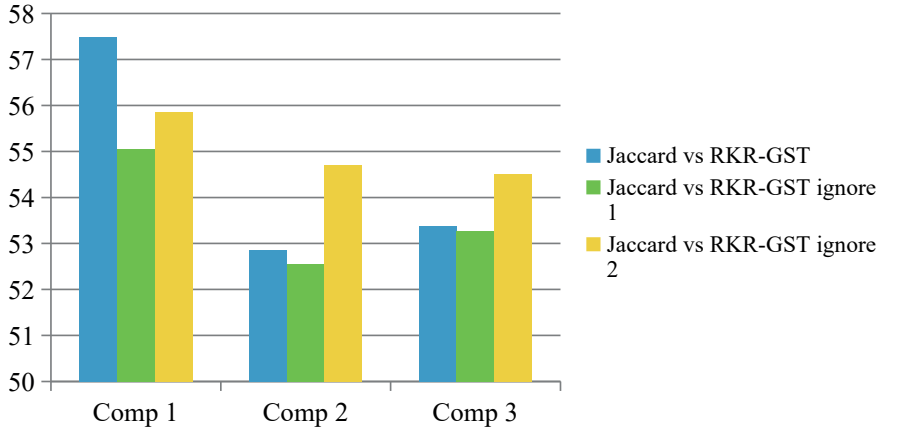


Figure 3. Changing of Jaccard and RKR-GST algorithms S-scores
Source: own study.

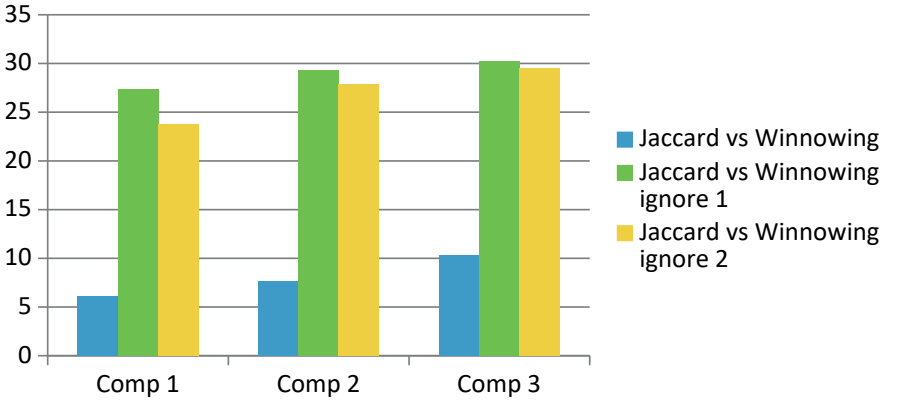


Figure 4. Changing of Jaccard and Winnowing algorithms S-scores
Source: own study.

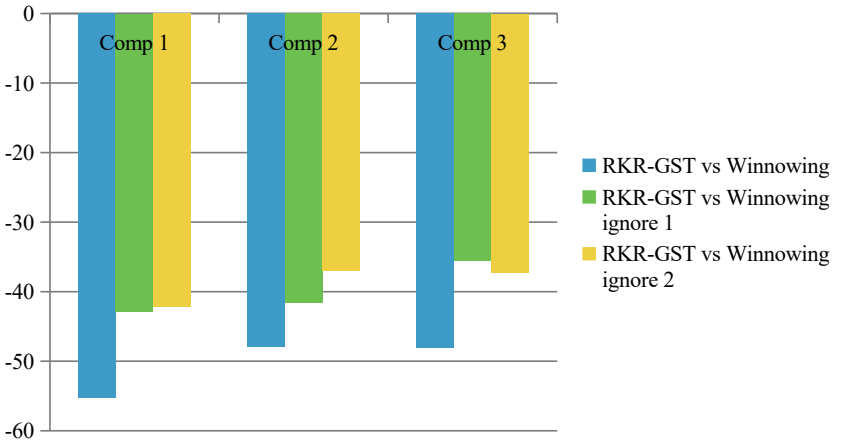


Figure 5. Changing of RKR-GST and Winnowing algorithms S-scores
Source: own study.

Discussion

Based on the results of the experimental investigation, objective estimates of the effectiveness of the Jaccard, RKR-GST, and Winnowing algorithms in a given software and hardware environment, with given input data, were obtained. The results confirm the suitability of these algorithms for use in analyzing the similarity of software projects for the purpose of detecting clones.

During the experiments, it was found that the Jaccard algorithm is the fastest, Winnowing is slightly slower (the difference ranges from 6-11 percent), and RKR-GST turned out to be the least efficient in terms of time, which is due to the complexity of its implementation and multi-level processing. This relationship is present in all series of experiments, including those where there is additional pre-processing of the program text database.

The change in the characteristics of the processors, namely the increase in cache performance while reducing the total frequency of the cores, allowed to improve the time efficiency of the Jaccard and to decrease Winnowing efficiency.

In cases of text preprocessing, a significant change in the efficiency indicators of the Jaccard and RKR-GST algorithm pairs, and RKR-GST and Winnowing on all hardware platforms is observed. In the first case, the efficiency increase is in favor of the Jacquard algorithm, in the second – RKR-GST.

During the development of application possible areas of improvement were also identified. In particular, it is advisable to implement the ability to load the base code that is not taken into account in the analysis, which would improve the accuracy of the results.

For the Winnowing algorithm, support for assessing similarity by Jaccard, Dice, or Andberg measures can be introduced, which would provide flexibility in the relationship by increasing the number of settings available to the user. For the RKR-GST algorithm, it is also advisable to implement additional evaluation methods – in particular, not only the average, but also the maximum similarity between pairs of texts.

Another important direction for future improvement of the application for analyzing is to increase performance. In the process of working with a large volume of projects and increasing the number of repetitions, a noticeable decrease in performance was found in the experiment. In view of this, it is advisable to implement support for multithreading or asynchronous execution to optimize analysis time and reduce the load on the system.

Conclusion

In the course of the work, a study of the time efficiency of algorithms for detecting software clones in a given software and hardware environment was conducted. Thanks to the use of SR-evaluation, not only the absolute performance indicators of the algorithms were determined, but also their pairwise comparison was performed, which allows us to give a numerical assessment of the advantages of one algorithm over another. The obtained estimates can be used as additional criteria when choosing algorithms for detecting clones in refactoring tasks, detecting code vulnerabilities or borrowings in the academic environment.

A software application was created for conducting experiments – the program clone analyzer. Its use in various software and hardware environments will further allow us to more accurately assess the impact of the environment on the efficiency of the algorithms and to formulate recommendations for the use of algorithms in specific conditions.

References

1. Ain, Q. U., Butt, W. H., Anwar, M. W., Azam, F., Maqbool, B. (2019). A systematic review on code clone detection. *IEEE Access*, *7*, 86121-86144.
2. Alon, U., Zilberstein, M., Levy, O., Yahav, E. (2018). *Code2vec: Learning distributed representations of code.*
3. Faisal, M., Nugroho, F., El Sulthan, M., Amini, F., Hariyadi, M. A., Sedayu, A. (2020). Plagiarism detection using manber and winnowing algorithm. *International Journal of Advanced Science and Technology*, *29*(6), 2130-2136.
4. Fletcher, S., Zahidul Islam, I. (2018). Comparing sets of patterns with the Jaccard index. *Australasian Journal of Information Systems*, *22*. https://doi.org/10.3127/ajis.v22i0.1538
5. Jo, Y.-B., Lee, J., Yoo, C.-J. (2021). Two-Pass technique for clone detection and type classification using tree-based convolution neural network. *Applied Sciences, 11*(14), 6613. https://doi.org/10.3390/app11146613
6. Martinez-Gil, J. (2024). Source Code Clone Detection Using Unsupervised Similarity Measures. In P. Bludau, R. Ramler, D. Winkler, J. Bergsmann (ed.), *Software Quality as a Foundation for Security*. SWQD 2024. Lecture Notes in Business Information Processing, vol 505. Springer. https://doi.org/10.1007/978-3-031-56281-5_2
7. Pravorska, N. I., Barmac, O. B., Medzatiy, D. M. (2021). The process of detecting blocks with repetitions and excess building using a language-independent increase detector. *Bulletin of Khmelnytskyi National University, 3,* 39-45.
8. Schleimer, S., Wilkerson, D. S., Aiken, A. (2003). *Winnowing: Local algorithms for document fingerprinting.* In I. Zachary, *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (pp. 76-85). Association for Computing Machinery. https://doi.org/10.1145/872757.872770
9. Shobha, G., Rana, A., Kansal, V., Tanwar, S. (2020). Code clone detection – A systematic review. In A. E. Hassanien, S. Bhattacharyya, S. Chakrabati, A. Bhattacharya, S. Dutta, (ed.), *Emerging Technologies in Data Mining and Information Security* (pp. 645-655). Springer.
10. Shrestha, S., Gautam, S., Sharma, K., Bhandari, A. (2023). Winnowing algorithm: A powerful tool for identifying plagiarism in assignments. *Journal of Trends in Computer Science and Smart Technology, 5*(2), 168-189. https://doi.org/10.36548/jtcsst.2023.2.006

11. Shynkarenko, V. (2009). *Experimental researches of algorithms in software and hardware environments*. Dnipropetrovsk National University of Railway Transport named after V. Lazaian.
12. Shynkarenko, V., Kuropiatnyk, O., Demidovich, I. (2025). Program clones detection as natural language texts fragments based on constructive-synthesizing modeling. In *CLW-2025: Computational Linguistics Workshop at 9th International Conference on Computational Linguistics and Intelligent Systems (CoLInS-2025), May 15-16, 2025, Kharkiv, Ukraine*. https://doi.org/10.31110/COLINS/2025-1/002
13. Walker, A., Cerny, T., Song, E. (2020). Open-source tools and benchmarks for code-clone detection: past, present, and future trends. *ACM SIGAPP Applied Computing Review, 19*(4), 28-39. https://doi.org/10.1145/3381307.3381310
14. Wang, Y., Ye, Y., Zhang, W., Xue, Y., Liu, Y. (2023). Comparison and evaluation of clone detection techniques with different code representations. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia*. https://doi.org/10.1109/ICSE48619.2023.00039
15. Wise, M. (1993). *String similarity via greedy string tiling and running karp-rabin matching*. Department of Computer Science Technical Report, Sydney University.
16. Zamrii, I., Kulakov, O. (2024). Overview of ways to automate the refactoring of code clones. *Connectivity*, *5*. https://doi.org/10.31673/2412-9070.2024.050575
17. Zhang, H., Sakurai, K. (2021). A survey of software clone detection from security perspective. *IEEE Access*, *9*, 48157-48173.

Netography

1. Aiken, A. (2022). *A system for detecting software similarity*. Retrieved from: https://theory.stanford.edu/~aiken/moss/ (access date: 10.09.2025).
2. Saglam, T. (1996). *JPlag – plagiarism detection tool*. Retrieved from: https://helmholtz.software/software/jplag (access date: 10.09.2025).
3. Yang, D. (2019). *Blog about MOSS*. Retrieved from: https://yangdanny97.github.io/blog/2019/05/03/MOSS (access date: 10.09.2025).