Detection of Programming Plagiarism in Computing Education: A Systematic Mapping Study

Kaio Pablo Gomes¹, Simone Nasser Matos²

¹Department of Informatics – Federal University of Technology – Paraná (UTFPR)
Ponta Grossa - PR, Brazil.

²Department of Informatics – Federal University of Technology – Paraná (UTFPR)
Ponta Grossa - PR, Brazil.

kgomes@alunos.utfpr.edu.br, snasser@utfpr.edu.br

Abstract. The programming plagiarism is increasingly a problem in computing education, and the proposed solutions for this growing concern rely on automatic detectors. The usage of the automatic tools for this purpose can provide benefits in education for professors and instructors of programming assignments, besides, to avoid the lack of essential skills from the students since they compromise their programming logic by plagiarizing. This paper performs a systematic mapping study aligned with a snowballing technique to analyzes the existing solutions for this domain. As contributions, tendencies, as well as information analysis, are provided to guide new proposals of solutions.

1. Introduction

Plagiarism in computing education, specifically in programming, is a problem that compromises critical skills for the development of programming logic by the students [Gomes and Matos 2019]. This misconduct occurs when a person copies either an entire program or a piece of source code made by other people [Andrianov et al. 2020]. As a solution for this academic dishonest, the usage of automatic detectors can reduce plagiarism in student’s assignments. Besides, the detection tools can reduce time consumption from professors’ grading and increase precision in the plagiarism analysis task [Gomes and Matos 2019].

Specific approaches were created to deal with programming plagiarism by elaborating solutions such as JPLAG, MOSS, Sherlock, SIM, YAP3, and Plaggie. For example, some of the main techniques for this purpose are tokens, hashes, common substrings, and signatures (fingerprints) [Allyson et al. 2018]. As shown in [Xu et al. 2020], these solutions for plagiarism detection still are limited by the demand for practical application. New opportunities and challenges generate the need for different approaches in order to accomplish other requirements. For example, “the capability to detect partial plagiarism,” “the resiliency to advanced code obfuscation,” “the interpretability of detection results,” and “the scalability to process large-scale software.”

Through a systematic mapping study (SMS) [Varela et al. 2017], the tendencies and insights about the existing models of plagiarism detection in programming can be analyzed. In order to extend the coverage for proposed detectors, a review technique such as snowballing [Wohlin 2014] assists in the search for complementary references. In this paper, an SMS with the snowballing technique is performed to evaluate solutions that deal with plagiarism in programming, which can be used for computing education.
2. Methodology

The SMS is a specific study that analyzes the findings in the literature for a subject of interest [Varela et al. 2017]. The SMS model adopted in this paper is based on the study shown in [Rattan et al. 2013] with modifications related to choosing each analyzed research. A flowchart illustrated in Figure 1 shows the functioning of the performed study, which can be abstracted into eight stages.

![Flowchart](image)

**Figure 1. Stages adopted in the systematic mapping study**

The first stage establishes SMS conduction guidelines, for example, the coverage period that was defined to contemplate from 2013 to 2018, and the subject to be investigated. From the chosen topic, stage 2 determines the study motivations for creating research questions. The main goals of this SMS were analysis the automatic detectors of plagiarism in programming by identifying their approaches, evaluating processes, supporting languages.

The following three research questions were created: 1) What are the approaches used to identify plagiarism in source code? 2) What plagiarism detection approaches have been evaluated through tests, and how many tests have been conducted? 3) What programming languages are supported in plagiarism detection approaches?

Step 3 considers which sources of information are used for searches. The IEEE Xplore and ACM DL were chosen since they are two of the most relevant in computer science topics, as shown in [Buchinger et al. 2014]. The search strings used in these sources of information represent stage 4, which was elaborated according to Table 1.

From the search results, a selection procedure in stage 5 defines the inclusion and exclusion criteria for choosing each research. The inclusion factors are the type of paper that only were considered those from conferences and journals, and the period of publication as defined in stage 1. The primary factor of exclusion was the presence of duplicated papers in addition to factors based on titles and abstracts out of scope. Altogether, at the end of the selection procedure, approximately 65.97% of the included studies were eliminated by the exclusion criteria.

**Table 1. Search strings used in each source of information**

<table>
<thead>
<tr>
<th>Search String</th>
<th>Source of Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ((Source*) OR (Software*) OR (Pro-gra)m*) AND (Plagia<em>r</em>)</td>
<td>IEEE Xplore e ACM DL</td>
</tr>
</tbody>
</table>

The quality evaluation (stage 6) seeks to rank the found studies as a form of identifying the most relevant works in the SMS. The adopted criterion was ordered by the...
number of citations, as shown in [Pagani et al. 2015]. After performing the classification, it was identified that 18.18% of the studies have not yet obtained citations. The average citation rate that each work has in this rank is five citations. However, a percentage of 76.32% of citations are concentrated among the top ten.

The data extraction in stage 7 identifies the following information for each research: title, authors, year of publication, type of paper, evaluation process, supported language, and approach. It started at the exclusion phase of the selection process in this paper. The last stage analysis every research in order to answer the three questions elaborated in stage 2. The obtained results are discussed in the next chapter.

3. Results


As a result of the first research question, 26 different approaches were identified. Figure 2 shows the usage frequency of approaches. Note that different works use the same techniques, with the token the most used among all identifications. However, approximately 46.15% of the approaches had the lowest frequency, indicating a lack of further validations. From the functioning of each solution, it was possible to perceive a trend associated with the usage of approaches with more than one technique. Altogether there were 31 studies, which represent 93.94% of the solutions. There was one technique only in two studies representing 6.06% out of all.

![Graph]. Answer for Research Question 1

**Figure 2. Frequency of the identified approaches**
Regarding the research question 2, Figure 3 shows the relation of frequency and number of source codes used in tests. It was found that 96.97% of the works used tests to validate the obtained results, while only one (3.03%) of them [Pohuda et al. 2014] proposed a solution without specifying the evaluation process. Points out that in [Roopam and Singh 2018] did not specify the number of source codes and was considered at least a test for each different repository.

![Answer for Research Question 2](image)

**Figure 3. The number of source codes used in tests and its frequency**

For the evaluation of the approaches, less than 100 source codes for testing were presented by 60.60% of the studies. The second highest frequency was the usage of between 100 and 1000 source codes, which was adopted by 30.30% out of all. The least frequent process used more than 1000 source codes, and it was chosen by 9.10% of the works.

It not common to share a source code dataset used for testing, only in the research made by [Mirza et al. 2017] adopted this type of resource for the evaluation process. The other works created their tests by developing plagiarism models implemented with the collaboration of students from computing courses. Another identified way of creating tests was generating either automatically or manually samples.

As an answer to research question 3, eight different programming languages were chosen to be supported for the solutions, as shown in Figure 4. However, 39.39% of the studies created independent approaches that do not limit the support feature. These studies show the importance of dealing with different languages, and it has surged a new tendency of using flexible solutions for supporting any source code.

![Answer for Research Question 3](image)

**Figure 4. Frequency of supported programming languages**
Among the solutions with a limited number of supported programming languages, the usage of Java stands out by representing 11 of 33 studies followed by the C language that is found in 10 of 33. It is identified that these types of dependent approaches tend to support more than one language.

4. Snowballing

The SMS accomplished in this paper was passed through an evaluation to certify whether there were changes in its obtained results considering a most recent coverage period. The snowballing applied to this paper was responsible for contemplating other studies between 2019 and 2020 in order to aggregate content with analysis and updates.

The snowballing technique strictly followed every stage in the SMS again except the two sources of information. Only the most relevant source in the computing between both was considered, which was the IEEE Xplore, as pointed by [Buchinger et al. 2014]. In all, 11 new studies presented in Table 2 were identified as candidates for the SMS.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Approach</th>
<th>Number of source codes</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kim et al. 2019]</td>
<td>Ordered Labeled Tree and LCS algorithm</td>
<td>768</td>
<td>Language-Independent for Android Apps</td>
</tr>
<tr>
<td>[Sun et al. 2019]</td>
<td>Bidirectional Static Slicing and Similarity Measures</td>
<td>619</td>
<td>Java</td>
</tr>
<tr>
<td>[Herrera et al. 2019]</td>
<td>Metrics</td>
<td>12426</td>
<td>Language-Independent</td>
</tr>
<tr>
<td>[Cheers et al. 2019]</td>
<td>Logic</td>
<td>5</td>
<td>Java</td>
</tr>
<tr>
<td>[Ljubovic and Pajic 2020]</td>
<td>Machine Learning and Metrics</td>
<td>11388</td>
<td>Language-Independent</td>
</tr>
<tr>
<td>[Andrianov et al. 2020]</td>
<td>Sparse Suffix Trees and binary mapping</td>
<td>Not mentioned</td>
<td>Language-Independent</td>
</tr>
<tr>
<td>[França et al. 2019]</td>
<td>Normalization and Sherlock N-overlap algorithm</td>
<td>2226</td>
<td>C and Java</td>
</tr>
</tbody>
</table>

By analyzing the results from the applied snowballing, it was noted that most trends identified in the SMS were confirmed in these new studies. Regarding the research question 1, approximately 81.82% of the solutions were based on combined techniques, and only 18.18% used a single type. There were ten different techniques found in the studies in relation to the first SMS, three of which introduced the artificial intelligence
field as a resource for three approaches. Nearly 42.30% of the techniques used in the SMS were reused in nine of the 11 new studies, and the most used approach was based on metrics rather than tokens.

For the new findings from research question 2, it was found that 100% of the works had an evaluation process. However, two of them did not mention the number of source codes used as tests. Among those who mentioned, approximately 88.88% exceeded the most common range of source code quantity in the SMS.

Regarding the supported programming languages, approximately 54.55% of the studies are developing a language-independent approach reiterating to be a desired feature. One of these solutions is aimed only at Android Apps; therefore, it covers any implementation language for this type of application. Still reiterating analyzes from the SMS, for those language-dependent solutions, Java was the most used followed by C and C++.

5. Concluding Remarks

This paper carried out an SMS to analyze the proposed solutions for the programming plagiarism problem in computing education. The applied methodology aligned with a snowballing technique aimed to identify three research questions from the most relevant studies from 2013 to 2020. The questions looked for retrieve information from the approaches, tests, and supported languages of the solutions.

Regarding the identified approaches, it was found the occurring of varied solutions represented by the usage of 36 different techniques in 44 studies. Also, this analysis indicated the presence of combined techniques in approximately 90.90% of the proposals. The evaluation process of these solutions did not occur only in one case, and three studies did not mention the number of source codes for tests. About the highest frequency of source codes used in experiments, it was less than 100 in the first SMS, and more than 100 for studies from 2019 and 2020.

A trend feature was observed by analyzing the research question 3. The feature is the creation of language-independent approaches found in 39.39% of the first SMS and 54.55% on the studies through snowballing. Among those language-dependent solutions, it was identified eight different supported programming languages: C, C++, Java, Assembly, Perl, Python, Php, and JavaScript. The most used language for supporting was Java, followed by C.

As future work, additional research questions can be added to the applied SMS as well as changes in its stages, such as covering a more extended period. For example, it is suggested to investigate whether there is a standard dataset for the evaluation process of the studies to provide conditions for comparing to each other. Besides, the identification of the performance metrics obtained by the solutions such as recall, precision, and f-measure can complement the evaluation process analysis.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.
References


Pham, M. T. and Nguyen, T. B. (2019). The domjudge based online judge system with plagiarism detection. In 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF), pages 1–6. IEEE.


