Contributions of Bioinformatics for computing education in the detection of programming assignment plagiarism

Kaio P. Gomes[1], **Simone N. Matos**[2]

[1]Department of Informatics
Federal University of Technology - Parana (UTFPR) – Ponta Grossa, PR – Brazil

[2]Department of Informatics
Federal University of Technology - Parana (UTFPR) – Ponta Grossa, PR – Brazil

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

Abstract. *Any source code can be modified in several ways to confuse plagiarism detection systems. Such diverse modifications require the usage of systems which can handle different types of plagiarism. The usage of automatic source code plagiarism detectors has implications for computing education. This paper extends Pedersen's work, a Bioinformatics method, by performing the application of this method on programming plagiarism domain, and by analyzing the usage of such tools through a discussion associated with the support for professors in assessing students' assignments. The application results are compared to a commonly used solution for the same purpose, the JPLAG tool. As a result of the evaluating study, the applied method showed a higher rate of similarity for specific types of plagiarism. Also, as a result of the analysis involving the use of an automatic tool for plagiarism in programming showed the benefits for computing education.*

1. Introduction

Plagiarism is a common problem that cannot be limited to academic cases [Chuda et al. 2012]. A survey with academic students indicates a rate of 72,5% from the participants admitted to plagiarizing at least once during their study obligations, for instance: writing a computing assignment or writing an essay, term paper, research, and others. Considering the particular case of writing a coding assignment, the results indicate that 50% of the students have plagiarized in programming courses by changing their source code [Sraka and Kaucic 2009].

Given the importance of the topic, plagiarism detection systems have been developed to assess student assignment authenticity [Prechelt et al. 2002]. As shown in [Le Nguyen et al. 2013], the usage of automatic plagiarism detectors can reduce considerably the academic dishonesty. Furthermore, such solutions can support professors by avoiding time-consuming and extra effort activities associated with the manual process of identifying plagiarism on student's assignment. However, the benefits are viable since the tools have high quality to assess plagiarism. Otherwise, the students are motivated to complain about false detections.

The programming plagiarism can be done by performing modifications on the source codes. For instance, modification of control structures, variables, data structures or structural redesign [Đurić and Gašević 2013]. According to [Roy and Cordy 2007], such

DOI: 10.5753/cbie.sbie.2019.1351

modifications can be classified into four types of source code plagiarism based on the level of identification difficulty. The identification of source code plagiarism is based on different methods and techniques that are applied to plagiarism detectors. Although there are a number of proposed solutions in computing plagiarism field, none of them might contemplate every type of source code plagiarism. However, a study in field of computer security, conducted by [Pedersen et al. 2012] has shown the opportunity of applying a Bioinformatics inspired method in the plagiarism detection area.

The Bioinformatics method proposed by [Pedersen et al. 2012] "utilizes an interdisciplinary approach to determine the pedigree of a digital artifact". This method models digital artifacts into a DNA sequence. For example, a source code file is a digital artifact since it has a binary representation and; for this reason, a process of turning such representations into a DNA sequence allows to execute powerful bioinformatics alignment tools to identify the level of similarity between these modeled sequences. The reporting generated from the utilized alignment tool shows the regions of similarities of these sequences, and it can be designed to provide information about the level of similarity among the digital artifacts or source codes. The similarity reporting can be used to build an automatic student plagiarism detection tool, which aids teachers to assess programming assignments in computing or related courses.

In the present work, we propose validating this Bioinformatics method through its application in the context of detecting source codes plagiarism and analyzing the implications of its usage for computing education. To perform the validation, we conduct an empirical study based on the execution of the Bioinformatics method in four different experiment scenarios. Each scenario is related to a specific type of source code plagiarism proposed by [Roy and Cordy 2007]. Every result is compared to one of the most accepted tools for detecting programming plagiarism, in this case, the chosen is the JPLAG.

2. Background

The application of the Bioinformatics method for detecting source code plagiarism requires knowledge over an interdisciplinary approach. It involves the field of Bioinformatics and programming plagiarism. This section shows basic concepts for both areas.

2.1. Bioinformatics

The deoxyribonucleic acid, commonly referred to as DNA, is related to the material that chromosomes are made of. The DNA is composed of a chain of repeating monomers, better known as nucleotides. The nucleotides found in deoxyribonucleic are four: guanine, cytosine, thymine, and adenine. These nucleotides are represented through the abbreviations G, C, T, A and U, respectively [Hunter 2012]. In the field of computing, the DNA can be modeled as a chain of characters [Arya et al. 2017]. The molecular structure of DNA is a double helix, which generates a double-stranded form [Watson et al. 1953]. Each strand is located by an associated direction, known as prime positions. The prime positions are referred to as 5' prime and 3' prime. The direction of the strands is from 5 to 3 or 3 to 5, each strand is anti-parallel to the other [Pedersen et al. 2012].

A DNA sequence can be compared to other DNA sequence through the process of alignment [Orabi et al. 2014]. The sequence alignment is a technique to identify regions of similarities among sequences. Figure 1 illustrates the process of alignment given two

DNA sequences. The sequence alignment can be done by two different approaches such as local alignment and global alignment. The global alignment recognizes the similarity between prime positions considering the entire sequences. Unlike global alignment, the local type seeks for specific regions of similarity. Both types of alignment shows how similar two sequences might be [Jayapriya and Arock 2015].

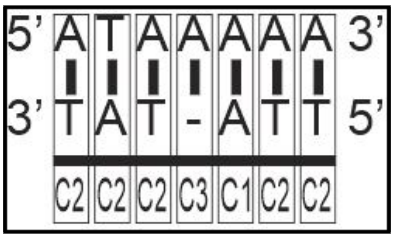

Figure 1. Example of alignment

As shown in Figure 1 , the DNA sequence alignment consists of comparing nucleotide by nucleotide. There are three scenarios as results of these comparisons: match, mismatch, and gap. The match scenario is the scenario when both nucleotides are equal. Unlike the match case, the mismatch scenario is when both nucleotides are different to each other. Finally, the gap scenario represents the absence of at least one nucleotide. A general process of alignment utilizes a scoring system. Each scoring system attributes different values for each possibility of existed scenario. The final score is the sum of all individual scores obtained in the alignment, and that gives the result of this general process.

2.2. Programming Plagiarism

According to [Parker and Hamblen 1989], a plagiarized source code can be defined as a result of modifications produced from another source code. For instance, the following modifications might be performed in a plagiarized source code: comments changes, indentations changes, control structures changes, variables changes, and data structures changes.

A plagiarized source code can be identified by its type of programming plagiarism as demonstrated by [Roy and Cordy 2007], and there are four types of programming plagiarism and two categories. The types are referred to as type I, type II, type III, and type IV. The categories based on similarity are textual and functional.

The type I are similar source codes except for modifications in its comments and whitespace. The Figure 2 shows an example of two fragments of codes modificated by comments and whitespace changes from type I. This type of plagiarism belongs to the textual category since the modifications are textual and not functional.

The type II, illustrated by the Figure 2, stands for modification on variables such as name changes, type changes, and literal changes. Furthermore, it might contain the type I as well.

For the type III, source codes can suffer further modification such as insertion of statements or a removal of statements. The Type I and II are also included. An example of type III is shown in the Figure 2.

Finally, the last type is the IV. It belongs to the functional similarity category since the source codes might be different on the textual content of the codes, but their functionalities are the same. It is the most difficult type to identify. The Figure 2 shows an example of this type.

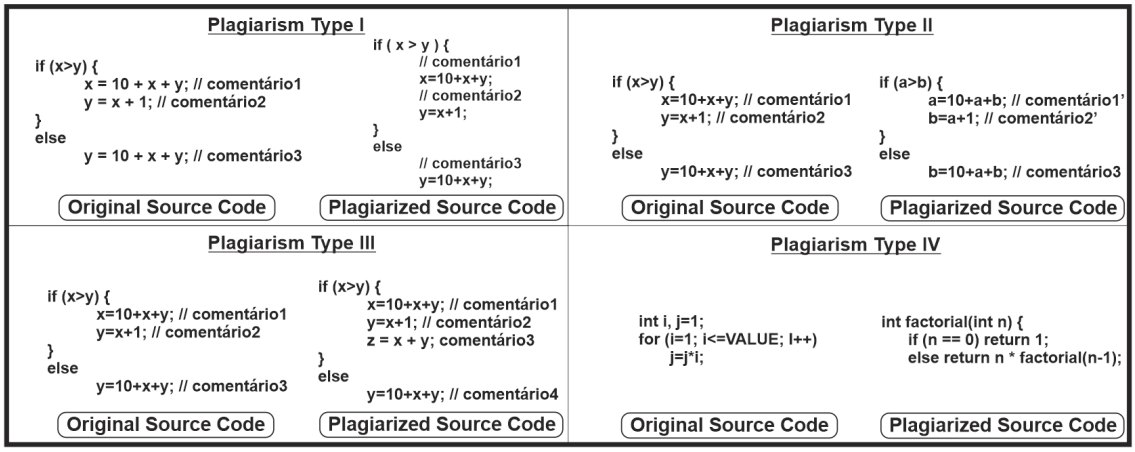

Figure 2. Examples of Source Code Plagiarism Types

There are other studies related to classifying the types of programming plagiarism as presented in [Sraka and Kaucic 2009]. However, in this presented work, the next section utilizes the plagiarism types that have been shown in this background.

3. Evaluating Study

Our evaluating study is based on four different scenarios. Each scenario seeks to evaluate the bioinformatics method for detecting a specific type of programming plagiarism, which is presented in the background section of this paper. This evaluating study aimed to answer two research questions:

- RQ1: Does the bioinformatics method identify the four specific types of source code plagiarism?
- RQ2: Does the source code plagiarism detection with the bioinformatics method have higher rate of similarity than JPLAG?

The bioinformatics method proposed by [Pedersen et al. 2012] can be abstracted in 4 main sequential steps when applied to source code plagiarism area. The general overview of the method is shown as an activity diagram in Figure 3. The step 1 is selecting two source codes as digital artifacts. Step 2 is generating two synthetic DNA from the digital artifacts. Step 3 is executing the alignment between two synthetic DNA. Finally, step 4 is identifying the rate of similarity from the executed sequence alignment.

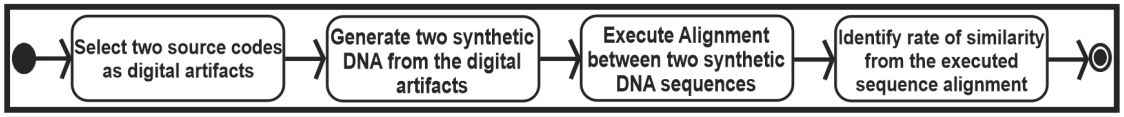

Figure 3. Bio-inspired method abstracted in four steps

In step 1, it is collected the source codes for detecting programming plagiarism. A source code is a digital artifact, which is an input for generating a synthetic DNA

sequence in the immediate next step. The step 2 is responsible for creating a synthetic DNA sequence from these artifacts.

A synthetic DNA sequence is a chain of nucleotides as well as a regular DNA sequence. However, it represents digital artifacts content and not living organisms. This creation is made with a specific algorithm developed by [Pedersen et al. 2012]. All the content of a digital artifact has its representation in ASCII binary code. For this reason, every textual aspect of a source code, which is the actual content, has a representation in binary. The size of each character in ASCII has one-byte size. A nucleotide is generated by two contiguous bits following a specific mapping. The mapping of this method defines the base "T" for bits "00", "G" for bits "01", "C" for bits "10" and "A" for bits "11". Thus, the algorithm works by mapping a character into four nucleotides as shown an example: the word "if" in Figure4.

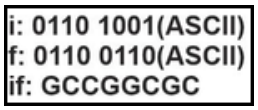

Figure 4. Mapping bits into nucleotides

In step 3, it is performed a sequence alignment for identifying similarities between two synthetic DNA sequences. This process is made by a bioinformatics tool. In this work, it is utilized the EMBOSS tool, even though the BLAST tool is a widely used bioinformatics tool as shown in [Sawyer et al. 2019]. The preference for EMBOSS was justified by the type of alignment, which is global. The BLAST tool is used for local alignment, and this evaluating study uses a global alignment. The bioinformatics tools provide a reporting about the alignment. This reporting is analyzed in the final step.

The last step, it analysis the reporting provided by the EMBOSS tool. One of the parameters of the alignment result is the rate of similarity. This numerical similarity value represents how similar two sequences are to each other. The goal is to identify a percentage value which is capable of showing how similar two source codes might be. For this reason, this parameter fits our needs in this application of the bioinformatics method for detecting programming plagiarism. Therefore, the result of the detecting plagiarism process, in this work, is based on the numerical value represented by the rate of similarity parameter.

3.1. Test Scenarios

This evaluating study executed 20 times this bioinformatics method for detecting programming plagiarism through 10 different source codes pairs as shown in Figure 5. Each scenario contemplates a specific type of plagiarism in source code. This evaluating study considers, in terms of analysis, just one test by scenario since the selected test can represent all related tests of the programming plagiarism type in question.

The selected tests for the four scenarios were extracted from the background section of this research. The source codes pairs are represented by the Figure 2. The scenarios I, II, III and IV represent plagiarism type I, II, III and IV, respectively. The next section discusses the findings after performed the application of the method.

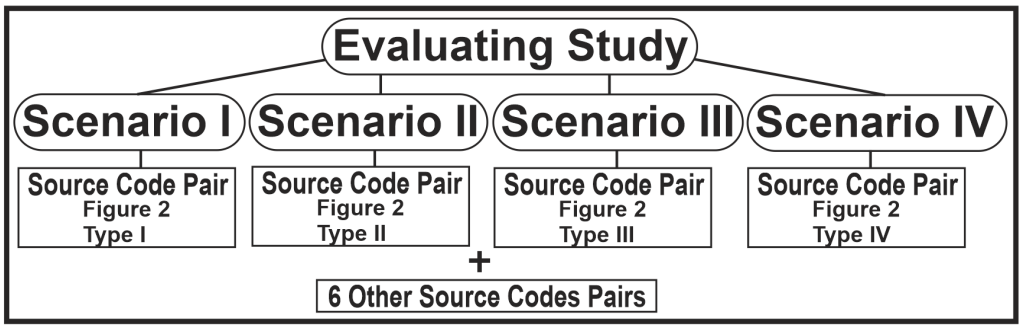

Figure 5. Source codes pairs from set of tests

4. Results

In table 1 is shown the results of this evaluating study. Each result assesses a specific type of programming plagiarism, which is identified by a rate of similarity in percentage. This numerical value is obtained from the bioinformatics method and the JPLAG.

Table 1. Evaluating Study Results

Test Scenario	Results		
	Type of Plagiarism	*Evaluated Method(%)*	*jplag(%)*
Scenario I	Type I	72.6	90-100
Scenario II	Type II	94.9	90-100
Scenario III	Type III	80.5	0-10
Scenario IV	Type IV	49.8	0-10

As proposed in the previous section, this study aimed to answer the following research questions:

4.1. RQ1: Does the bioinformatics method identify the four specific types of source code plagiarism?

For The type I, the evaluated method shows a rate of 72.6% of similarity between the selected two source codes in scenario I. This result indicates the compromised ability of this method to recognize precisely techniques related to variations in comments and whitespace. This problem is justified by the feature of mapping every character of the source code content. For this reason, whitespaces and comments are also mapped into a nucleotide even though they should be irrelevant for the detecting process.

Although the method is still capable of recognizing programming plagiarism type I as we can see at its rate of similarity, it could be improved by using source code normalization filters. The use of a filtering process can eliminate unnecessary mapping of bits into nucleotides. For instance, the comments or whitespaces could be ignored before mapping a source code.

The evaluated method is able of recognizing the type II as shown in table I. The rate of similarity is 94.9%, which indicates the precise recognition for this specific programming plagiarism type. The techniques of changing variables are not sufficient to confuse this method in the process of recognizing this type of plagiarism.

In the plagiarism type 3, the evaluated method shows a rate of 80.5%. This result ensures the potential ability of this method in recognizing further modification such as insertion of statements or a removal of statements. Also, it demonstrates its efficiency to handle type I and II at the same time the plagiarized source code utilizes techniques of further modifications.

Lastly, the type 4 which is the most difficult plagiarism type according to [Roy and Cordy 2007], the rate of similarity is 49.8%. This result indicates that changes need to be addressed for improving the ability of the evaluated method for detecting this type of plagiarism.

4.2. RQ2: Does source code detection with the bioinformatics method have higher rate of similarity than JPLAG?

The result presented by the evaluated method for the plagiarism type I is 72.6%. However, JPLAG identifies the same plagiarism type with a rate between 90% and 100%. In this scenario, the JPLAG has a higher rate of similarity than the evaluated method.

For the scenario II, the evaluated method has a rate of 94.9%. The JPLAG identifies a similarity range between 90% and 100%. Both plagiarism detection systems show the same result, but the evaluated method is more specific.

While JPLAG has not identified plagiarism as expected for scenario III, the evaluated method recognizes a rate of 80.5%. The plagiarism type III is related to further modification and not just basic techniques for confusing plagiarism detection systems as the type I and II. The JPLAG considers the source code pair as being the lowest similarity range, which is from 0% to 10%.

In the last scenario, the presented results are different for each plagiarism detection system. While the JPLAG has considered the lowest similarity range, the evaluated method identifies a rate of 49.8% in similarity for the source code pair tested.

5. Implications of Bioinformatics for Computing Education

Nowadays, different studies have been proposed to innovate the educational scenario associated with teaching-learning processes. According to [Barbosa and Souza 2018], the usage of software and automatic tools is a trend in the evolution of educational processes. The support for pedagogical decision-making is only one of the potential applications aided by using technology. This way, a Bioinformatics inspired method application for helping professors with the problem of detecting source code plagiarism in student assignments from programming or related courses is useful. The benefits of such solutions can be associated with the reduction of time-consuming and error-prone activities.

A solution based on bioinformatics is considered for solving the plagiarism detection since this interdisciplinary field of science has been studying similarity problem involving DNA and protein sequences. As a result, several efficient methods have been developed to compute how similar different sequences can be. By transforming source code into biological sequences, it is possible to seize such tools to apply an efficient method to detect programming plagiarism. The expected results are the possibility to detect different types of plagiarism with a less computational cost. To confirm such an ideal goal, this present work showed the application of a bioinformatics method to detect every type of source code plagiarism following the classification indicated by [Roy and Cordy 2007].

To seek for plagiarism in individual assignments from different students can be a complex task, which consumes extra effort. An automatic tool for this purpose comes in handy to avoid such problems aside from the elimination or reduction of academic dishonesty that affects teaching-learning processes. However, the proposed tool has to be capable of detecting as much as possible source code plagiarism types considering the field of programming. Otherwise, the benefits cannot be viable since the quality of the solution is compromised. As suggested for [Almeida et al. 2018], the indicators to assess the quality of educational software is a form to validate a proposed solution such a detector of plagiarism. One of the indicators is associated with the presence of failures, by which in this context would demotivate professors. Furthermore, the students would complain about the results of a low-quality programming plagiarism detector.

Once found a way to handle the programming plagiarism problem in education, the professors can act to avoid and identify the motivations and reasons why their students are committing plagiarism. Not to mention the fact that along the academic journey of students, the awareness of legal and ethical aspects involved in plagiarism could be increased as mentioned by [Mozgovoy et al. 2010] the importance of tackling it.

As pointed out by [Ribeiro et al. 2018b, Stadelhofer and Gasparini 2018], most of the students have difficulty to learn programming, and others have no motivation because they consider it as an irrelevant theme. Such thoughts contribute to raising academic dishonesty on assignments and other obligations during a computing course. Additionally, the advent of the internet allows anyone to find a potential solution for his academic assignments by simply copying someone else's work [Novak 2016].

To address the problem of difficulty, the usage of an automatic detector by professors can be useful to identify potential students passing through this issue, and it is possible to make an appropriate decision regarding to support those who are not understanding a subject covered in the plagiarized assignment. Likewise, it is helpful to use such tool to prevent a student from failing a class since the same identified students are likely to fail as stated by [Ribeiro et al. 2018a] a study about the importance to early predicts the potential failing students.

6. Concluding Remarks

This research aimed to verify the applicability of the bioinformatics method for detecting source codes plagiarism, and its implications for computing education. According to the evaluating study, the bioinformatics method shows its potential to contemplate all the four specific types of source code plagiarism following the classification proposed by [Roy and Cordy 2007]. In addition, the method has presented a higher rate of similarity than JPLAG in specific scenarios of plagiarized source codes.

The proposed method has the potential to become an automatic tool for detecting source code plagiarism. Such a tool can be useful to support professors in computing and related courses by seeking plagiarism in students assignments and other class obligations involving coding. The implications for computing education seems to be diverse, and the direct benefits are the reduction of time-consuming and extra effort activities for professors. Furthermore, the indirect benefits are the identification of potential failing students, students with difficulty to learn programming and contribution to the formation of students with legal and ethical awareness regarding academic dishonesty.

As future work, we intend to propose a novel approach to detect source code plagiarism based on the bioinformatics method applied in the present work. The main goal of a novel method is to overcome the weakness of the evaluated method. Also, we intend to evaluate the implications for computing education in a real-world scenario in university courses based on empirical research.

Acknowledgment

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

References

Almeida, A., Gomes, T., Leal, V., Gomes, R., and Leal, L. (2018). Indicadores para avaliação de software educacional com base no guia gdsm (goal driven software measurement). In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 29, page 21.

Arya, G. P., Bharti, R. K., Prasad, D., and Garg, V. (2017). An improved method for dna sequence compression. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pages 1–4.

Barbosa, R. and Souza, R. (2018). Um levantamento dos determinantes de inovação em softwares educacionais. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 29, page 437.

Chuda, D., Navrat, P., Kovacova, B., and Humay, P. (2012). The issue of (software) plagiarism: A student view. *IEEE Transactions on Education*, 55(1):22–28.

Hunter, L. E. (2012). *The processes of life: an introduction to molecular biology*. Mit Press.

Jayapriya, J. and Arock, M. (2015). Pairwise local alignment using wavelet transform. In *2015 Annual IEEE India Conference (INDICON)*, pages 1–5.

Le Nguyen, T. T., Carbone, A., Sheard, J., and Schuhmacher, M. (2013). Integrating source code plagiarism into a virtual learning environment: benefits for students and staff. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, pages 155–164. Australian Computer Society, Inc.

Mozgovoy, M., Kakkonen, T., and Cosma, G. (2010). Automatic student plagiarism detection: future perspectives. *Journal of Educational Computing Research*, 43(4):511–531.

Novak, M. (2016). Review of source-code plagiarism detection in academia. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 796–801. IEEE.

Orabi, E. S., Assal, M. A., Azim, M. A., and Kamal, Y. (2014). Dna fingerprint using smith waterman algorithm by grid computing. In *2014 9th International Conference on Informatics and Systems*, pages PDC–74–PDC–79.

Parker, A. and Hamblen, J. O. (1989). Computer algorithms for plagiarism detection. *IEEE Transactions on Education*, 32(2):94–99.

Pedersen, J., Bastola, D., Dick, K., Gandhi, R., and Mahoney, W. (2012). Blast your way through malware malware analysis assisted by bioinformatics tools. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Prechelt, L., Malpohl, G., and Philippsen, M. (2002). Finding plagiarisms among a set of programs with jplag. *J. UCS*, 8(11):1016.

Ribeiro, M., Paes, R., Neto, B. S., Pereira, J. L., Castro, T., and Gheyi, R. (2018a). 30 days after introducing programming: Which of my students are likely to fail? In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 29, page 1283.

Ribeiro, R. B., Fernandes, D., de Carvalho, L. S. G., and Oliveira, E. (2018b). Gamificação de um sistema de juiz online para motivar alunos em disciplina de programação introdutória. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 29, page 805.

Roy, C. K. and Cordy, J. R. (2007). A survey on software clone detection research. *Queen's School of Computing TR*, 541(115):64–68.

Sawyer, S., Horton, M., Burdyshaw, C., Brook, G., and Rekapalli, B. (2019). Hpc-blast: Distributed blast for modern hpc clusters. In *Proceedings of 11th International Conference*, volume 60, pages 1–14.

Sraka, D. and Kaucic, B. (2009). Source code plagiarism. In *Information Technology Interfaces, 2009. ITI'09. Proceedings of the ITI 2009 31st International Conference on*, pages 461–466. IEEE.

Stadelhofer, L. E. and Gasparini, I. (2018). Ensino de algoritmos e lógica de programação para os diferentes cursos: Um mapeamento sistemático da literatura. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 29, page 108.

Watson, J. D., Crick, F. H., et al. (1953). Molecular structure of nucleic acids. *Nature*, 171(4356):737–738.

Đurić, Z. and Gašević, D. (2013). A source code similarity system for plagiarism detection. *The Computer Journal*, 56(1):70–86.