Journal of Information Technology and Computer Science
Volume 6, Number 1, April 2021, pp. 9-17
Journal Homepage: www.jitecs.ub.ac.id

Initial Suspicion on Detecting Code Plagiarism and Collusion in Academia: Case Study of Algorithm and Data Structure Courses

Mewati Ayub*[1], Oscar Karnalim[2], Maresha Caroline Wijanto[3], Risal[4]

[1,2,3,4]Faculty of Information Technology, Maranatha Christian University, Bandung, Indonesia

{[1]mewati.ayub@it.maranatha.edu, [2]oscar.karnalim@it.maranatha.edu, [3]maresha.cw@it.maranatha.edu, [4]laurentius.risal@it.maranatha.edu}

*Corresponding Author

Received 30 October 2020; accepted 26 March 2021

Abstract. In engineering education, some assignments require the students to submit program code, and since that code might be a result of plagiarism or collusion, a similarity detection tool is often used to filter excessively similar programs. To improve the scalability of such a tool, it is suggested to initially suspect some programs and only compare those programs to others (instead of exhaustively compare all programs to one another). This paper compares the effectiveness of two common techniques to raise such initial suspicion: focusing on the submissions of smart students (as they are likely to be copied), or the submissions of slow-paced students (since those students are likely to breach academic integrity to get higher assignment mark). Our study shows that the latter statistically outperforms the former by 13% in terms of precision; slow-paced students are likely to be the perpetrators, but they fail to get the submissions of smart students.

Keyword : students, SPS, perpetrators, precision

1 Introduction

Academic dishonesty is a concerning issue in engineering education [1], [2], especially when the classes are going online due to Covid-19 pandemic [3]. In online environment, assignments are completed without direct supervision by the lecturer and this might tempt the students to cheat [4]. A study even shows that student submissions during the pandemic have higher similarity than the ones before the pandemic [5]. It becomes worse as some of the assignments are not text-based and therefore cannot be easily detected with Turnitin [6], a popular text-based similarity detection system. Programming courses offered in some engineering majors (e.g., information technology, electrical engineering, and mechatronics) for instance, assess the students based on their submitted program code, which is arguably different from standard text [7].

To maintain academic integrity in programming, many attempts have been proposed and generally, they can be classified into five categories [8]. The most obvious attempt is educating students about academic integrity and penalizing them if any breaches of academic integrity occur. Sometimes, it is also possible to make

cheating difficult by introducing additional restrictions during completion of the assignment (e.g., requiring students to do the work during a particular period of time while directly monitored by lecturers or tutors). Alternatively, lecturers can also reduce the benefits of cheating by giving many assignments with a small portion to the overall mark.

Considering student psychological states, lecturers can discourage cheating by informing them about the real consequences if they are cheating. They can also encourage students to maintain academic integrity by providing additional help like peer-assisted learning.

In programming courses, plagiarism and collusion are often found as attempts of breaching academic integrity. Code plagiarism refers to illegally reusing program code with no or limited acknowledgment to the authors [9]. Code collusion is quite similar to code plagiarism except that the original authors are aware of such misbehavior and let it happens [10].

To detect code plagiarism and collusion, a similarity detection tool like JPlag [11] is often used to raise suspicion of programs with unduly similarity. That kind of tool typically works in twofold [12]. Each student submission is translated to an intermediate representation like source code tokens [13], linearized syntax tree [14], and low-level tokens [15]. All submissions are subsequently compared to one another and submission pairs with overly high similarity are reported. The similarity algorithms vary from cosine similarity from Information Retrieval [16], running Karp-Rabin greedy string tiling [17] from string matching, to string alignment [18].

Many existing similarity detection tools are not really scalable due to cubic time complexity of their pairwise comparison, especially when a large number of long student submissions are involved. As a result, it is suggested to initially suspect some submissions based on the lecturer's knowledge about the students, and only compare those submissions to others [19]. This positively affects the scalability given that the time complexity becomes linear to the number of initially suspected submissions. Further, this might increase the accuracy as well since the excluded submissions are unlikely to be relevant.

Nevertheless, to the best of our knowledge, no studies specifically evaluate how the initial suspicion is raised, though it directly affects the effectiveness. An existing study about a scalable similarity detection tool [19] is only focused on the development of the tool and evaluating other aspects instead of how initial suspicion is raised. This paper compares two popular ways of raising initial suspicion. The first one is to use the submissions of smart students, assuming that these submissions are the main target of the perpetrators since they tend to result in high assignment mark. The second one is to use the submissions of slow-paced students, assuming that the students might be the perpetrators of academic dishonesty. As our case study, the comparison was performed on three programming classes for one academic semester with a total of 67 assignments and 1034 student submissions.

It is worth to note that this study does not contribute a new similarity detection for code plagiarism and collusion. Instead, it empirically compares two ways of using scalable similarity detection in three algorithm and data structure classes.

The paper is organized as follows: Section 2 describes our method evaluating the impact of initial suspicion; Section 3 shows our results and discussion; Section 4 explains our limitations; and finally Section 5 is about conclusions and future work.

2 Method

The study empirically compares the effectiveness of two techniques of raising initial

suspicion. The first one initially suspects the submissions of smart students as those submissions are likely to be copied by the perpetrators who want to get a high assessment mark. The second one does the opposite by using the submissions of slow-paced students; some of those students might be the perpetrators since academic dishonesty often occurs as a result of desperate act seeking help [20] for higher assignment mark. For convenience, the former will be referred to as smart-oriented suspicion (SS) while the latter will be referred to as slow-paced oriented suspicion (SPS).

Both techniques of raising initial suspicion require lecturer's knowledge about student academic performance. We cannot therefore use publicly available data sets [21], [22], [23], [24] as they have no such information. We create our own data set just for this purpose.

As our case study, the comparison was performed on three classes of algorithm and data structure; two of them covered basic materials (BC) while another one covered the ad-vanced ones (AC). In total there are 1034 student submissions collected from 41 weekly student assignments for one academic semester (14 weeks). Table 1 shows statistics of the average, maximum, and minimum score of each class. On average, their scores are not much different.

Table 1 Statistics

Class	Number of students	Average score	Maximum score	Minimum score
BC1	29	86.43	100	24.14
BC2	17	79.63	98.07	3.21
AC	9	80.2	92.1	70.2

BC contributes 28 weekly assignments as it has two classes (BC1 and BC2) in which each of them has 14 weekly assignments. The assignments are about implementing the algorithm and data structure taught in the lecture with Python as the programming language. Some weekly assignments consist of more than one task. In terms of class size, the first class has 29 students enrolled while the second one only has 17 students.

AC has 13 Java and one C# weekly assignments. Since our search of copied programs was based on a similarity detection tool that does not cover C# (will be explained later), only the Java assignments are considered. They are about implementing algorithm and data structure and learning new programming languages. Similar to BC, the assignments occasionally have more than one task. AC has one class with nine students enrolled.

Our method works as in Figure 1. At first, student code files were collected for all assignments. For each class, the lecturer was asked to find copied programs of each weekly assignment with both suspicion techniques, via the help of a similarity detection tool proposed in [19].

The tool [19] searches similar code files in the same assignment based on the given code file. It has four modes. The first one is to convert the code file to token string and then measure the similarity with RKRGST, which is partly adapted from JPlag [11]. The second one is similar to the first except that RKRGST is replaced with cosine similarity from Information Retrieval, which is more time-efficient but less accurate. The third one is derived from the second, but the token string is taken from a linearized syntax tree, that is believed to be more effective though the translation takes a considerable amount of time. The fourth one uses a linearized syntax tree but with cosine similarity to compensate for its long execution time. Further details about the tool can be found in the following paper [19].

In this case, we set the tool to use the second mode, token string with cosine similarity since it is arguably the most scalable one. We do not use several similarity measurements as our main goal is to evaluate the effectiveness of common techniques of raising initial suspicion, not the similarity measurements.

Per initially suspected submission, the tool compared it with other submissions in which top-5 program pairs with the highest similarity were used as the basis of performance comparison. A suspicion technique is more effective than another if its suggested program pairs result in higher effectiveness.

Precision, a metric from information retrieval [25] was used to represent the effectiveness. It is the proportion of suggested program pairs that are actually copied (true positives) to all suggested program pairs (true + false positives). Typically, this metric is featured with recall, which is the proportion of suggested program pairs (true positives) that are actually copied to all copied program pairs (true positives + false negatives). However, since the determination of copied program pairs is based on the suggested program pairs, that metric is not applicable as the result will be the same to precision; both will have the same divider. If there were more than one initially suspected submission, the precision would be averaged prior comparison. The formula of precision is given in (1) and recall in (2).

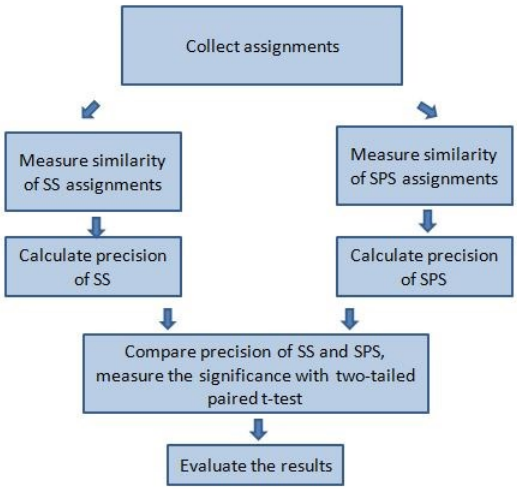

Fig. 1. Research method diagram

$$Precision = \frac{true\_positives}{true\_positives + false\_positives} \quad (1)$$

$$Recall = \frac{true\_positives}{true\_positives + false\_negatives} \quad (2)$$

Since the assignments are arguably simple, the resulted precision is not expected to be high. However, we believe this is not an issue in our study as our goal is to compare the suspicion techniques, not to propose an effective technique.

3 Result and Discussion

Table 2 shows that both techniques result in zero precision in half of BC1 assessments. Further observation shows that these assignments only had one expected solution with limited possible variation; most of them were given in early lecture weeks and they were simple by default. Moreover, there was no convincing evidence for plagiarism and collusion on their suggested program pairs (e.g., uncommon identifier names or whitespaces).

More BC1 assignments favor SPS than SS; the former outperforms the latter in five of seven remaining assignments (week 6, 7, 8, 12, and 14). The perpetrators were mostly slow-paced students, but on most occasions, they copied the code from average-paced or other slow-paced students. The smart students were aware of the consequences of letting their code copied by the perpetrators and decided to keep their code only for their own.

In general, SS and SPS are equally effective. Their averaged precision scores are similar (12% and 14% respectively) while the difference is only about 2%. When measured with two-tailed paired t-test with 95% confidence rate, the difference is not significant (p-value = 0.6).

Table 2 Resulted precision for BC1

Week	SS	SPS	SPS - SS
1	0%	0%	0%
2	0%	0%	0%
3	0%	0%	0%
4	0%	0%	0%
5	0%	0%	0%
6	40%	60%	20%
7	30%	40%	10%
8	0%	40%	40%
9	0%	0%	0%
10	0%	0%	0%
11	20%	0%	-20%
12	0%	20%	20%
13	40%	0%	-40%
14	36%	40%	4%
Average	**12%**	**14%**	**2%**

For BC2 assignments, fewer assignments result in zero precision for both techniques at once. Table 3 shows only week 1 and week 9 fall to that category, though the assignments were similar to those given in BC1. Further observation shows that the perpetrators in this class left many convincing pieces of evidence for plagiarism and collusion, even for simple tasks. The copied programs either have uncommon variable names, logic flows, or whitespaces.

Among the remaining twelve assignments, nine of them favor SPS over SS while two of them see both as equally effective. Similar to BC1, the perpetrators were slow-paced students who did not copy code from smart students. They mainly got the code from average or slow-paced students. The smart students were good in not sharing their code to others.

SPS is more effective than SS since its average precision (47%) is 13% higher. However, the difference is still not statistically significant, though the p-value is lower than that of BC1 (p-value = 0.1).

The effectiveness of suspicion techniques might be affected by the students' behavior. BC2 results in higher averaged precision than BC1 for both techniques despite they share the same assignment tasks. Further, it has fewer zero precision

scores and favors SPS more. We observe two potential reasons. First, BC2 perpetrators exclusively copied code with unusual patterns and those patterns were useful for raising suspicion. Second, BC2 had more perpetrators than BC1 and most of them were recognized by the lecturer as slow-paced students.

Table 3 Resulted precision for BC2

Week	SS	SPS	SPS - SS
1	0%	0%	0%
2	0%	40%	40%
3	40%	40%	0%
4	60%	65%	5%
5	80%	80%	0%
6	100%	50%	-50%
7	80%	85%	5%
8	80%	100%	20%
9	0%	0%	0%
10	0%	20%	20%
11	0%	40%	40%
12	40%	50%	10%
13	0%	20%	20%
14	0%	73%	73%
Average	**34%**	**47%**	**13%**

Table 4 shows that no AC assignments result in zero precision for both suspicion techniques at once. In other words, at least one of our proposed techniques is guaranteed to suggest copied programs, which is promising. Similar to BC2, many of the perpetrators left convincing evidence (primarily uncommon identifier names), and those were really useful for raising suspicion of plagiarism and collusion.

Among 14 assignments, ten of them favor SPS, one favors SS, two favor both equally, and one is not applicable for this study (week 13) given the assignment was completed in C#, a programming language that is not covered by our tool. SPS is more effective than SS as the averaged precision is about two times higher (47%) and the difference is statistically significant when measured with two-tailed paired t-test with 95% confidence rate (p-value = 0.01). The perpetrators were mostly slow-paced students and they copied the programs from average-paced or slow-paced students instead of the smart ones.

Table 4 Resulted precision for AC

Week	SS	SPS	SPS-SS
1	20%	20%	0%
2	0%	20%	20%
3	0%	20%	20%
4	40%	20%	-20%
5	20%	27%	7%
6	20%	44%	24%
7	100%	100%	0%
8	60%	70%	10%
9	0%	33%	33%
10	0%	73%	73%
11	0%	80%	80%
12	0%	52%	52%
13	NA	NA	NA
14	40%	50%	10%

Average	**23%**	**47%**	**24%**

When all assignments are considered, SS results in 23% averaged precision while SPS results in 36%. SPS is generally more effective than SS since most of the perpetrators were slow-paced students and they failed to copy the code from smart students. The difference is statistically significant as the p-value is lower than 0.01 when measured with two-tailed paired t-test with 95% confidence rate.

4 Limitations of the Study

Our study has three limitations. First, we only considered three algorithm and data structure classes. It is possible that the findings might be changed when more classes are considered and/or the study was performed on other programming courses like object-oriented programming. Second, the similarity measurement used in our study is cosine similarity with token strings. The findings can be different if the similarity measurement is replaced with other algorithms like running Karp-Rabin greedy string tiling [17], and/or it utilizes more advanced intermediate representations like syntax tree [14]. Third, per initially suspected submission, only top-five program pairs with the highest similarity are considered. More program pairs involved might change the findings as they affect the resulted precision.

5 Conclusion and Future Work

To improve the scalability of code plagiarism and collusion detection, it is arguably important not to compare all student programs to one another. Consequently, a study suggests to initially suspect some programs and only compare those programs to others. This paper compares the effectiveness of two common techniques of raising initial suspicion of plagiarism and collusion. The first one focuses on the programs of smart students, which are likely to be copied if the perpetrators aim to get high assignment marks. The second one focuses on the programs of slow-paced students since those students might breach academic integrity to gain higher assignment marks.

According to the study, future use of scalable similarity detection is expected to focus on slow-paced students. That results in higher effectiveness since these students are likely to be the perpetrators and they copy the programs from average-paced or other slow-paced students. They might want to get the programs from smart students but apparently, those smart students are smart to keep their programs for themselves.

It is worth noting that both techniques of raising initial suspicion do not show high precision. Focusing on smart students only shows 23% on average while focusing on slow-paced students shows 36%. This is due to the fact that the assignments are arguably simple, and it is quite difficult to differentiate evident similarity from the coincidental one.

For future work, we plan to integrate student performance to a code similarity detection system so that it can automatically list the slow-paced students and compare their programs to others. We also plan to reconduct this study on assignments from other programming courses to check the consistency of our findings.

6 Acknowledgment

This research has been supported by a research grant provided by Maranatha Christian University, Indonesia. The authors would like to thank Gisela Kurniawati

and Rossevine Artha Nathasya from Maranatha Christian University for their participation in this study.

References

1. Metruk, R.: Confronting the Challenges of MALL: Distraction, Cheating, and Teacher Readiness. International Journal of Emerging Technologies in Learning (iJET), 15(2), 4–14 (2020).
2. Halak, B. and El-Hajjar, M.: Plagiarism detection and prevention techniques in engineering education. In: 11th European Workshop on Microelectronics Education, pp. 1–3, Southhampton (2016).
3. Masterson,M. : An Exploration of the Potential Role of Digital Technologies for Promoting Learning in Foreign Language Classrooms: Lessons for a Pandemic. International Journal of Emerging Technologies in Learning (iJET), 15(14) , 83–96 (2020).
4. McCabe, D.L., Treviño, L.K., and Butterfield, K.D. : Cheating in academic institutions: a decade of research. Ethics & Behavior, 11(3), 219–232 (2001).
5. Karnalim, O., Simon, Ayub, M., Kurniawati, G., Nathasya, R. A. and Wijanto, M. C.: Work-in-Progress: Syntactic Code Similarity Detection in Strongly Directed Assignments. In 2021 IEEE Global Engineering Education Conference (EDUCON), (2021).
6. Batane, T.: Turning to Turnitin to Fight Plagiarism among University Students. Journal of Educational Technology & Society, 13(2), 1–12 (2010).
7. Simon, Cook, B., Sheard, J., Carbone, A., and Johnson, C. : Academic integrity: differences between computing assignments and essays. In: 13th Koli Calling International Conference on Computing Education Research, pp. 23–32, (2013).
8. Sheard, J., Simon, Butler, M., Falkner, K., Morgan, M. and Weerasinghe, A.: Strategies for Maintaining Academic Integrity in First-Year Computing Courses. In ITiCSE '17: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 244-249, (2017).
9. Cosma, G. and Joy, M.: Towards a definition of source-code plagiarism. IEEE Transactions on Education, 51(2), 195–200 (2008).
10. Fraser, R.: Collaboration, collusion and plagiarism in computer science coursework. Informatics in Education, 13(2), 179–195 (2014).
11. Prechelt, L., Malpohl, G., and Philippsen, M.: Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science, 8(11), 1016–1038 (2002).
12. Karnalim, O., Simon, and Chivers, W.: Similarity detection techniques for academic source code plagiarism and collusion: a review. In: IEEE International Conference on Teaching, Assignment, and Learning for Engineering (TALE), (2019).
13. Bejarano, A.M., García, L.E., and Zurek, E.E.: Detection of source code similitude in academic environments. Computer Applications in Engineering Education, 23(1), 13–22 (2015).
14. Wang, L., Jiang, L., and Qin, G.: A search of Verilog code plagiarism detection method. In: 13th International Conference on Computer Science & Education, pp. 1–5, (2018).
15. Karnalim, O.: IR-based technique for linearizing abstract method invocation in plagiarism-suspected source code pair. Journal of King Saud University - Computer and Information Sciences, 31(3), 327–334 (2019).
16. Flores, E., Barrón-Cedeño, A., Moreno, L. and Rosso, P.: Uncovering source code reuse in large-scale academic environments. Computer Applications in Engineering Education, 23(3), 383–390 (2015).
17. Wise, M.J.: YAP3: improved detection of similarities in computer program and other texts. In: 27th SIGCSE Technical Symposium on Computer Science Education, pp. 130–134, (1996).
18. Huang, X., Hardison, R.C., and Miller, W.: A space-efficient algorithm for local similarities. Bioinformatics, 6(4), 373–381 (1990).
19. Franclinton, R., Karnalim, O., and Ayub, M.: A Scalable Code Similarity Detection with

Online Architecture and Focused Comparison for Maintaining Academic Integrity in Programming. International Journal of Online and Biomedical Engineering (iJOE), 16(10), 40–52 (2020).
20. Vogts, D.: Plagiarising of Source Code by Novice Programmers a 'Cry for Help'? In: 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, pp. 141–149, (2009).
21. Heres, D. and Hage, J.: A Quantitative Comparison of Program Plagiarism Detection Tools. In CSERC '17: Proceedings of the 6th Computer Science Education Research Conference, pp. 73-82, (2017).
22. Karnalim, O., Budi, S., Toba, H., Joy, M.: Source Code Plagiarism Detection in Academia with Information Retrieval: Dataset and the Observation, Informatics in Education 18(2), pp. 321-344, (2019).
23. Mirza, O.M., Joy, M., and Cosma, G.: Style Analysis for Source Code Plagiarism Detection — An Analysis of a Dataset of Student Coursework. In 2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT), Timisoara, pp. 296-297, (2017).
24. Ljubovic, V., and Pajic, E.: Plagiarism Detection in Computer Programming Using Feature Extraction From Ultra-Fine-Grained Repositories. IEEE Access, 8, pp. 96505-96514, (2020).
25. Croft, W.B., Metzler, D., and Strohman, T.: Search Engines : Information Retrieval in Practice. Addison-Wesley (2010).