Selection of Code Segments for Exclusion from Code Similarity Detection

Simon*
University of Newcastle
Australia
simon@newcastle.edu.au

Oscar Karnalim*†
University of Newcastle
Australia
oscar.karnalim@uon.edu.au

Judy Sheard*
Monash University
Australia
judy.sheard@monash.edu

Ilir Dema
University of Toronto Mississauga
Canada
ilir.dema@mail.utoronto.ca

Amey Karkare
Indian Institute of Technology Kanpur
India
karkare@iitk.ac.in

Juho Leinonen
University of Helsinki
Finland
juho.leinonen@helsinki.fi

Michael Liut
University of Toronto Mississauga
Canada
michael.liut@utoronto.ca

Renée McCauley
College of Charleston
USA
mccauleyr@cofc.edu

ABSTRACT

When student programs are compared for similarity, certain segments of code are always sure to be similar. Some of these segments are boilerplate code – public static void main String [] args and the like – and some will be code that was provided to students as part of the assessment specification. The purpose of this working group is to explore what other code is expected to be reasonably common in student assessments, and should therefore be excluded from similarity checking. The answers will clearly vary with programming language, and perhaps with level of assessment item.

Working group members will collect assessment submissions from their own or their colleagues' students, and it is hoped that these submissions will together encompass a wide variety of assessment tasks in a wide variety of programming languages.

The working group aims to deliver clear guidelines as to what code can reasonably be excluded from automatic code similarity detection in various circumstances. It also aims to deliver a summary of what sort of code lecturers tend to provide for students when setting an assigned task, and why they provide that code.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

Plagiarism; collusion; academic integrity; code similarity detection

*Working group leader
†Also with Maranatha Christian University.

ITiCSE '20, June 15–19, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-6874-2/20/06.
https://doi.org/10.1145/3341525.3394987

ACM Reference Format:
Simon, Oscar Karnalim, Judy Sheard, Ilir Dema, Amey Karkare, Juho Leinonen, Michael Liut, and Renée McCauley. 2020. Selection of Code Segments for Exclusion from Code Similarity Detection. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20), June 15–19, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3341525.3394987

1 BACKGROUND

In the attempt to ascertain whether different student programs were written independently, some academics apply automatic code similarity detection tools to help detect whether two or more programs are more similar than one would expect from coincidence alone.

Mann and Frew [6] point out that, particularly in early programming courses, most programs to achieve the same task will have a great deal in common; that program similarity is not necessarily an indicator of program copying or unauthorised assistance. Nevertheless, particularly in large classes, some sort of automatic filtering of programs is almost essential to distinguish those programs that are clearly independent from those that merit further examination by a human to determine whether they have a common source.

Filtering techniques can be categorised into two groups based on how the common code segments are defined: manually or automatically. Manual techniques typically use a similarity algorithm to search code for predefined segments, which are then removed prior to comparison [2, 5, 7]. Such techniques have been applied in some popular code similarity detection tools such as JPlag[1] [8], MOSS[2] [9], Plaggie[3] [1], and Sherlock[4] [4]. With automatic techniques, common segments are determined on the basis of their distribution across the whole body of student assignments [3], the most frequently occurring segments being deemed as common. This can

[1]https://jplag.ipd.kit.edu/
[2]https://theory.stanford.edu/ aiken/moss/
[3]https://www.cs.hut.fi/Software/Plaggie/
[4]https://warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock/

be a good approach if it is not known what code the programs are likely to share.

Even though there are many filtering techniques available, the criteria for determining common code segments seem somewhat arbitrary. Further observation is needed to define what the criteria are and why they are needed.

2 METHOD

There is a great deal of literature on the application of code similarity detection to academic misconduct such as plagiarism and collusion in computer programming. The working group will examine that literature for papers that mention the removal of common code before programs are compared, and will collate any descriptions of exactly what code is removed.

Each member of the working group will gather programming assignments submitted by students in courses at their own institution. Members have all ascertained what process they are required to follow to gain ethics approval for the use of student submissions, and have applied for and attained such approval.

Members of the group will then use a code similarity detection tool to analyse the programs from their own institutions. This will help them to identify code that is common to many or all of the programs, and they will attempt to identify that code as standard, lecturer-provided, coincidentally common, etc. The expected deliverable from this analysis is a clear guideline as to what code can generally be removed from programs written in various programming languages before those programs are compared for inappropriate similarity. Application of this guideline in automatic code similarity detectors is likely to lead to more effective similarity detection by reducing the number of false positives caused by the necessarily common code.

The working group will conduct an informal survey of programming educators, asking them if they are willing to provide sample assessment specifications, along with explanations of their reasons for including any specific code.

For the student assignments that they are analysing, members will also examine the task specifications to determine whether code was provided to the students, and, if so, the nature of that code. Where possible, the person who set the task will be asked why they provided the code to students. The expected deliverable from this analysis is a summary of the sorts of code that people provide to their students when setting assignment tasks, and why they provide that code.

REFERENCES

[1] Aleksi Ahtiainen, Sami Surakka, and Mikko Rahikainen. 2006. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In *Sixth Baltic Sea conference on Computing Education Research, Koli Calling 2006*. ACM Press, Uppsala, 141–142. https://doi.org/10.1145/1315803.1315831

[2] Zoran Đurić and Dragan Gašević. 2013. A source code similarity system for plagiarism detection. *Computer Journal* 56, 1 (Jan 2013), 70–86. https://doi.org/10.1093/comjnl/bxs018

[3] Christian Domin, Henning Pohl, and Markus Krause. 2016. Improving plagiarism detection in coding assignments by dynamic removal of common ground. In *2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM Press, San Jose, 1173–1179. https://doi.org/10.1145/2851581.2892512

[4] Mike Joy and Michael Luck. 1999. Plagiarism in programming assignments. *IEEE Transactions on Education* 42, 2 (1999), 129–133. https://doi.org/10.1109/13.762946

[5] Dragutin Kermek and Matija Novak. 2016. Process model improvement for source code plagiarism detection in student programming assignments. *Informatics in Education* 15, 1 (2016), 103–126. https://doi.org/10.15388/infedu.2016.06

[6] Samuel Mann and Zelda Frew. 2006. Similarity and originality in code: plagiarism and normal variation in student assignments. In *Eighth Australasian Computing Education Conference*. Australian Computer Society, Inc, Hobart, 143–150.

[7] Jonathan YH Poon, Kazunari Sugiyama, Yee Fan Tan, and Min-Yen Kan. 2012. Instructor-centric source code plagiarism detection and plagiarism corpus. In *17th ACM Annual Conference on Innovation and Technology in Computer Science Education*. ACM Press, Haifa, 122. https://doi.org/10.1145/2325296.2325328

[8] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. 2002. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science* 8, 11 (2002), 1016–1038.

[9] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *2003 ACM International Conference on Management of Data*. ACM Press, San Diego, 76–85. https://doi.org/10.1145/872757.872770