Informatics in Education, 2023, Vol. 00, No. 00, –

DOI: 10.15388/infedu.2023.05
ACCEPTED VERSION (WITHOUT THE JOURNAL LAYOUT).

Identifying Plagiarised Programming Assignments with Detection Tool Consensus

Hayden Cheers [1], Yuqing Lin[1,2], and Weigen Yan [2]

[1] The University of Newcastle, Callaghan, NSW 2308, Australia
[2] Jimei University, Fujian, China

ARTICLE HISTORY
Compiled April 14, 2022

ABSTRACT
Source code plagiarism is a common occurrence in undergraduate computer science education. Many source code plagiarism detection tools have been proposed to address this problem. However, most of these tools only measure the similarity between assignment submissions, and do not actually identify which are suspicious of plagiarism. This work presents a semi-automatic approach that enables the indication of suspicious assignment submissions by analysing source code similarity scores among the submissions. The proposed approach seeks the consensus of multiple source code plagiarism detection tools in order to identify program pairs that are consistently evaluated with high similarity. A case study is presented to demonstrate the use of the proposed approach. The results of this case study indicate that it can accurately identify assignment submissions that are suspicious of plagiarism.

Keywords: source code plagiarism detection, behavioural similarity, source code similarity.

1. Introduction

Source code plagiarism is a common occurrence in undergraduate computer science courses. Prior studies have indicated between 50% to 79% of undergraduate students will plagiarise at least once during their studies [Curtis and Popal(2011), Pierce and Zilles(2017), Sraka and Kaucic(2009), Yeo(2007)]. With the recent impact of the COVID-19 pandemic, there have also been some reports of an increased prevalence of undergraduate plagiarism, especially in computer science courses. For example, the Australian Broadcasting Corporation reported in late 2020 that an entire class in a computer science course at the Australian National University was to be penalised 30% of their course mark due to a widespread suspicion of contract cheating [Evans(2020)], while [Lancaster and Codrin(2021)] also reported an increased use of the homework help website Chegg, that can be used by students to obtain help with or purchase answers to assignment questions. Whether or not such cases are representative of a widespread increase in plagiarism, plagiarism is a form of academic misconduct, and as such, cases of plagiarism in academic environments need to be identified and addressed no matter the perceived occurrence.

Corresponding Author: Yuqing Lin. Email: yuqing.lin@newcastle.edu.au
The third author is supported by NSFC Grant 12071180

When an academic is marking programming assignments, source code plagiarism can typically be suspected when work submitted for one assignment shares a large subset of source code with another [Cosma and Joy(2008)] (e.g. via copy and paste), or when one submission is copy or very similar of another [Parker and Hamblen(1989)]. In addition to examine all submissions for an assignment, very often, we have to consider identifying cases of source code plagiarism that are sourced from external sources (e.g. students in previous course offerings or from sources on the web). For example, many students can be seen to have archived their works on the public web in version control websites such as GitHub. This has led to a large search space for detecting source code plagiarism, overall making the detection of plagiarism a continually difficult and time-consuming process.

To aid in the identification of source code plagiarism, many automated *Source Code Plagiarism Detection Tools* (SCPDTs) [Martins *et al.*(2014), Novak *et al.*(2019)] have been proposed, also being referred to as source code similarity detection/measurement tools [Joy and Luck(1999)] or software/code plagiarism detection tools [Roy *et al.*(2009), Ragkhitwetsagul *et al.*(2016), Ragkhitwetsagul *et al.*(2018)]. Such tools evaluate the similarity scores of pairs of assignment submissions in a data set. This score commonly indicates what percentage of one submission can be found in another. A high similarity score indicates that two submissions share a large portion of code which are similar, and is considered suspicious of plagiarism. A mid-range score can imply that students might have colluded on an assignment (a form of academic misconduct) or simply be a result of students completing an assignment with similar application designs (e.g. as influenced by course work). A low score generally means the submissions are dis-similar.

Contrary to their name, SCPDTs do not detect plagiarism. SCPDTs can only be considered to identify *indications* of plagiarism [Joy and Luck(1999)]. These indications are represented through the similarity scores of the submissions, which are used by a human reviewer to judge whether plagiarism has occurred. While the reported similarity scores support the detection of plagiarism, there are no clear and unambiguous criteria to determine plagiarism based on the similarity scores. Consequently, an reviewer is required to manually inspect submission pairs with subjectively high similarity scores. This is a largely unassisted process, where the reviewer must use their own academic judgement to determine if any submissions are suspicious of plagiarism.

In an ideal scenario, there will be a clear distinction between the similarity scores of program pairs that *are* plagiarised, and those that *are not* plagiarised. For example, plagiarised pairs would always be reported as (or near) 100% similar, while innocent pairs will be reported with a low similarity (e.g. less than 20%). Hence, a human reviewer could then easily examine the obvious high-valued scores, and separate them from the low-valued scores. However, students are known to hide their plagiarism by modifying the source code of a program [Faidhi and Robinson(1987), Joy and Luck(1999), Cosma and Joy(2012), Cheers *et al.*(2020)]. Such plagiarism-hiding source code modifications are applied in an attempt to reduce the evaluated similarity of the plagiarised work, and hence it is common to see similarity scores of plagiarised programs that are considerably lower than 100%. When reviewing assignments and their similarity scores, it can then become difficult to identify program pairs that are suspicious of plagiarism. This then ultimately increases the workload of an academic.

In order to address this issue, this work proposes a semi-automated approach for indicating program pairs that are suspicious of plagiarism based on their similarity scores. The approach is based on the observation that an individual SCPDT provides

its own perspective of whether plagiarism is present between a program pair based on if it evaluates a high similarity score, relative to any other program pairs in a data set. This perspective is a result of how a SCPDT evaluates the similarity of program pairs, and this can result in different SCPDTs providing different indications of whether plagiarism is present [Ahadi and Mathieson(2019)]. The perspective of one SCPDT can never be correct or incorrect by itself without further investigation. Hence, in order to gain confidence of a SCPDT is indeed providing an indication of plagiarism, the perspective of not one but multiple SCPDTs are combined to find consensus in indicating cases of plagiarism. By combining the results of multiple SCPDTs and finding consensus amongst their indications of plagiarism, a semi-automated method of detecting plagiarism can be afforded. This approach is then intended to support a human reviewer in efficiently analysing the results of a SCPDT for indications of plagiarism, and to subsequently aid in the overall detection of plagiarism.

The remainder of this work is structured as followed. Section 2 presents background on methods of measuring source code similarity, commonly available SCPDTs and commonly used methods of identifying suspicious similarity scores. Section 3 describes the proposed consensus-based approach for the indication program pairs that are suspicious of plagiarism. Section 4 presents a case study detailing the use of the proposed approach. And finally, Section 5 discusses the results of the case study and using the proposed approach, and concludes this work.

2. Background

2.1. Types of Similarity

As stated, SCPDTs measure source code similarity. However, source code has intrinsic qualities that can be measured for similarity, and this has resulted in many different methods of measuring source code similarity. For the purpose of this work, methods of measuring source code similarity are generalised as being either: structural, semantic or behavioural. Each of which can be measured and compared to provide a perspective of how two pieces of source code are similar.

Structural similarity represents how similar the composition two source code files are by measuring their internal structures. In its most basic form, structural similarity can be measured with textual strings. This is by applying techniques such as string edit distance or string alignment to measure the similarity of source code [Joy and Luck(1999), Pike(n.d.), Gitchell and Tran(1999), Rani and Singh(2018)]. However, it is more common to see structural similarity measured with the comparison of lexical token sequences, representing the structure of the source code in terms of important lexical elements. Subsequently, structural similarity can be measured with token edit distance or tiling-based approaches [Joy and Luck(1999), Prechelt and Malpohl(2003), Schleimer *et al.*(2003)Schleimer, Wilkerson, and Aiken, Ahtiainen *et al.*(2006), Grune and Huntjens(1989), Anzai and Watanobe(2019)]. Other approaches measure the structural similarity of parse trees or abstract syntax trees, representing the source code within the grammar of a programming language [Li and Zhong(2010), Zhao *et al.*(2015), Fu *et al.*(2017)].

Semantic similarity represents the similarity in the meaning of source code. This is through semantic analysis, that analyses source code to extract information not expressed through the grammar of a programming language. Semantic approaches typically analyse a program through program dependence graphs

[Ferrante *et al.*(1987)]. The program dependence graph identifies the relations between terms within a procedure or method. Subsequently, the similarity of these graphs can be calculated (e.g. with graph edit distance or sub-graph embedding [Liu *et al.*(2006)Liu, Chen, Han, and Yu, Chen *et al.*(2010), Chae *et al.*(2013)]) to identify similar source code. Other semantic methods include applying latent semantic analysis to identify similarly referenced terms [Cosma and Joy(2012)] or identifying similar call graph structures [Prado *et al.*(2018)].

Behavioural similarity (also known as dynamic analysis) identifies similar runtime behaviours of source codes. There are a diverse range of techniques applied to identify behavioural similarity. This can be by analysing the functional equivalence of a program [Li *et al.*(2016), Bertran *et al.*(2005)], the use of data at runtime [Jhi *et al.*(2011)], identifying similar interactions with the execution environment [Anjali *et al.*(2015)], or identifying similar program logic [Zhang *et al.*(2014), Luo *et al.*(2017)]. Such approaches are based on the assumption that the behaviour of a program expressed through source code is a uniquely identifying feature, and that similar behaviours indicate similar fragments of source code.

2.2. Source Code Plagiarism Detection Tools

There are many SCPDTs proposed to aid in the identification of source code plagiarism [Joy and Luck(1999), Martins *et al.*(2014), Novak *et al.*(2019)]. Most SCPDTs follow the same basic principle of identifying indications of plagiarism by evaluating the similarity of two assignment submissions. This may be on a file-wise basis (where a tool reports similarity of all file pairs between two submissions), and/or on a submission-wise basis (where a tool reports the similarity of two individual assignment submissions).

A recent survey by [Novak *et al.*(2019)] indicated that there have been at least 170 SCPDTs proposed in recent years. However, despite the large number of proposed tools, the overwhelming majority are not made available for reuse after publication [Cheers *et al.*(2020), Novak *et al.*(2019)]. Subsequently, there exist six commonly used SCPDTs referenced in academic works:

- MOSS [Schleimer *et al.*(2003)Schleimer, Wilkerson, and Aiken]
- JPlag [Prechelt and Malpohl(2003)]
- Plaggie [Ahtiainen *et al.*(2006)]
- Sim [Grune and Huntjens(1989)]
- Sherlock-W [Joy and Luck(1999)]
- Sherlock-S [Pike(n.d.)]

MOSS [Schleimer *et al.*(2003)Schleimer, Wilkerson, and Aiken] evaluates source code similarity using a winnowing-based document fingerprinting algorithm. First, documents are pre-processed to remove irrelevant details (e.g. whitespace and formatting). Second, documents are divided into k-grams (continuous sub-strings of length k) which are then hashed. A sliding window is then used to identify a subset of hashes which form a fingerprint of the document. Similar documents are then found by comparing these fingerprints, with similar documents containing an overlap of fingerprints.

JPlag [Prechelt and Malpohl(2003)] operates by applying a token tiling algorithm to cover one source code file with tokens extracted from another. If two source files have a large degree of coverage, they can be considered similar and hence suspicious of plagiarism. First, source code files are converted into a stream of lexical tokens. Second, the extracted tokens are compared between files to determine similarity by

the Running-Karp-Rabin greedy string tiling algorithm [Karp and Rabin(1987)] where tokens from one file are covered over another within a tolerance of mis-match. Program similarity is evaluated as the percentage of tokens from one program which can be tiled over another.

Plaggie [Ahtiainen *et al.*(2006)] is a tool that is claimed to operate similarly to JPlag. However, it is an entirely local application, compared to JPlag which was originally provided as a web service. No known publication describes the operation of Plaggie. However from examining the implementation of Plaggie, it operates upon tokenised representations of source code evaluating similarity with token tiling.

Sim [Grune and Huntjens(1989)] analyses programs for structural similarity through the use of string alignment. For two programs, Sim will firstly parse the source code creating a parse tree. The tool will then represent the parse trees as strings and align them by inserting spaces to obtain a maximal common sub-sequence of their contained tokens. The similarity of programs is then evaluated as the quantity of matches.

Sherlock-W [Joy and Luck(1999)] implements both text and tokenised comparison methods. In the tool, a pair of programs are compared for similarity five times: in their original form, with whitespace removed, with comments removed, with whitespace and comments removed, and as a tokenised source file. In all cases, the comparisons measure similarity through the identification of common *runs*. A *run* is a sequence of lines shared between two files.

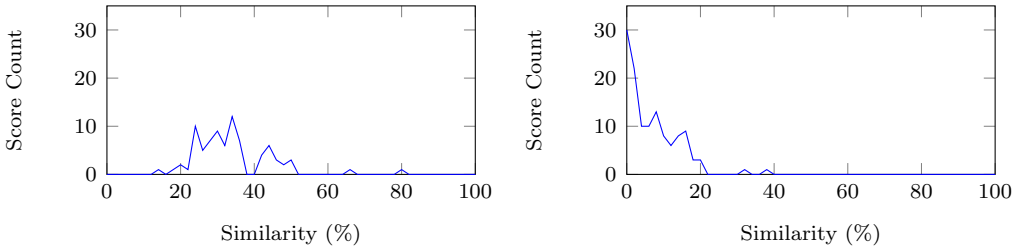
Sherlock-S [Pike(n.d.)] analyses programs by extracting digital signatures. Digital signatures of source code are generated by hashing string token sequences extracted from text files. The digital signatures are then compared, with the similarity of files being evaluated as the number of digital signatures in common.

These six common tools measure the structural similarity of source code. Unfortunately, there are few known or available approaches that measure the semantic or behavioural similarity of source code for source code plagiarism detection. One semantics-based approach is Plagate [Cosma and Joy(2012)]. Plagate applies latent semantic analysis to match source code documents with similar terms, and allows for identifying similar documents with similar terms used in similar manners. While Cheers, Lin and Smith [Cheers *et al.*(2021b)] utilised a Program Dependence Graph-based tool to evaluate semantic similarity for plagiarism detection. This tool is referred to as 'Graph ED', and measures similarity by identifying similar relations between terms in source code. BPlag [Cheers *et al.*(2021a)] is a recently proposed behavioural SCPDT that analyses the behavioural similarity of programs by identifying common execution behaviours through a process based on symbolic execution.

2.3. Identifying Suspicious Similarity Scores

As existing SCPDTs do not indicate suspicious similarity scores, prior works that evaluate SCPDTs often use fixed score cut-off values to identify suspicious similarity scores used to indicate the presence of plagiarism [Novak *et al.*(2019)]. For example, prior works have used two common types of cut-offs: scores above a threshold of $x\%$, or the top r ranked scores in a data set [Allyson *et al.*(2019), Cosma and Joy(2012), Durić and Gašević(2013), Ragkhitwetsagul *et al.*(2018), Zheng *et al.*(2018)].

When applying score cut-offs, often a single $x\%$ threshold or top r value is used to identify suspicious scores. Prechelt and Malpohl [Prechelt and Malpohl(2003)] applied a threshold of 50% to identify suspicious scores reported by JPlag. Similarly, Ramirez-



(a) Scores evaluated with JPlag.

(b) Scores evaluated with Sherlock-S.

Figure 1.: An example of variability in similarity score distribution for a single data set when evaluated with different SCPDTs. JPlag (left) places most scores in the 20% to 50% range, with outliers at approx. 70% and 80%. Sherlock-S (right) places most scores below 20%, with 2 outliers above 30%.

de-la Cruz et al. [Ramírez-de-la Cruz *et al.*(2014)] applied a threshold value of 50% to identify plagiarised scores. Cosma and Joy [Cosma and Joy(2012)] evaluated their tool, Plagate, against JPlag and Sherlock-W on four data sets of assignment submissions. In this evaluation, different threshold or cut-off values were utilised for each tool and evaluation data set. Cosma and Joy utilised score thresholds between 70% and 80% with Plagate, and 54.8% and 100% for JPlag; as well as applied a cut-off to the top r scores from Sherlock-W, with r values between 20 and 50. Burrows et al. [Burrows *et al.*(2007)] utilised a threshold value of 30% for use with JPlag, that was justified by knowing in advance the lowest similarity score in a data set. Zheng et al. [Zheng *et al.*(2018)] analysed the top ten results in evaluating their tool CodEX.

There also exist works that apply a sliding window of cut-offs to identify an optimal value. Allyson et al. [Allyson *et al.*(2019)] evaluated their tool, Sherlock N-Overlap, against Sherlock-S, JPlag, Sim and MOSS on 10 code samples. Utilising threshold values 10%, 20%, .., 90%; Allyson et al. identified optimal threshold ranges between 40% to 70% for Sim, 10% to 70% for MOSS, and 30% to 70% for JPlag. Similarly, Duric [Durić and Gašević(2013)] evaluated their tool, SCSDS, against JPlag with cut-off thresholds between 10% and 90% (at 5% steps). Subsequently, they identified SCDSD had optimal accuracy between thresholds of 35% to 50%, while JPlag had high accuracy between thresholds of 30% to 60%. Ragkhitwetsagul et al. [Ragkhitwetsagul *et al.*(2018)] applied threshold values of 19% for JPlag, 19% for Plaggie, 6% for Sherlock-S, and 16% for Sim in an evaluation of source code similarity. These values were identified in advance to produce the best possible results on their evaluation data sets.

The problem with these approaches is that, as the SCPDTs do not indicate what scores are suspicious, different cut-off values need to be identified ahead-of-time. This then results in a wide range of selected cut-off values being used for different SCPDTs and data sets. In experimental scenarios with pre-assessed evaluation data sets, an optimal x or r value can be identified ahead of time to identify known suspicious scores. However, in a real world scenario, a reviewer using a SCPDT will not know in advance what submissions are suspicious, or how similar any suspicious submissions will be in terms of similarity score. It also needs to be considered that there is great variability in the scores reported by individual SCPDTs for the same data set. Fig. 1 exemplifies this on a set of undergraduate assignment submissions. Sherlock-S (Fig. 1b) reports considerably lower-valued similarity scores than evaluated by JPlag (Fig. 1a). However, within the scores reported by both tools, two outliers can be identified: scores above

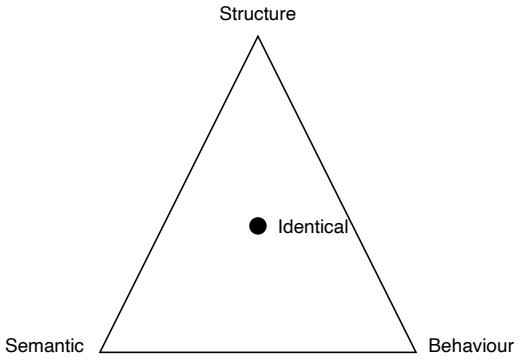


Figure 2.: The three aspects of source code similarity.

60% for JPlag, and scores above 30% for Sherlock-S. In both cases, the scores are clearly outliers in the data sets, and hence should be considered suspicious of plagiarism. This great variability results in a large range of threshold or top r cut-offs applied in prior studies. For example, consider the range of thresholds utilised for JPlag, that range between 19% [Ragkhitwetsagul *et al.*(2018)] and 100% [Cosma and Joy(2012)]. It is unlikely a human reviewer would consider the lower end of the range (i.e. 19%) as being suspicious of plagiarism and use that value in a real world evaluation. It is also difficult to conceive such values being re-applied between different data sets from different groups of students. For example, considering a first year introductory programming course at a university where the programs developed by students are simple and often developed by re-imagining some existing samples provided in the course. The similarity between such submissions is very likely to be higher than the similarity between assignment submissions of a third year programming course where students have much broader skill sets. Hence, pre-defined cut-off values alone are not appropriate to be used outside of experimental settings.

3. Proposed Approach

The proposed approach is underpinned by considering source code similarity to be a milti-dimensional value. Source code can be compared with different methods measuring different types of similarity, then a combined multi-dimensional representation of similarity can be derived. Each individual type of similairty (structural, semantic and behavioural) is considered an axis of similarity, each being referred to as an *'aspect of similarity'*. The similarity of two programs is then considered to be a value that encompasses these aspects, to express a more inclusive and broader interpretation of similarity. Fig. 2 provides a graphical overview of how two programs can be considered similar using these three aspects. The approach uses this interpretation of similarity to analyse the results of different SCPDTs in order to indicate program pairs that are suspicious of plagiarism. It achieves this by identifying consensus amongst the tools for program pairs that are consistently measured with high similarity values in all aspects.

To derive consensus amongst the SCPDTs for what program pairs appear suspicious of plagiarism, the approach first needs to determine what are high similarity scores. As mentioned, there are no unambiguous criteria for what is a high similarity score. Hence, a limiting criteria must be applied. For the purpose of this work, the threshold x% and

top r cut-offs are used. Following the identification of high similarity scores, consensus is identified as an intersection of the highest limited similarity scores reported by the SCPDTs. Two different intersections are used as part of this approach, each of which provides a different level of confidence in the scores they suggest. The first consensus criterion is referred to as the 'High Confidence' consensus criterion. It is identified as the intersection of scores reported by all three SCPDTs, calculated as $A \cap B \cap C$ (or shorthand $\cap 3$). This criterion requires that for a similarity score to be considered suspicious, it must be within the highest scores reported by each SCPDT. This ensures that all indicated program pairs are: highly structurally similar, highly semantically similar, and highly behaviourally similar. The second consensus criteria is referred to as the 'Broad' consensus criteria. It is defined as the intersection of scores as reported by any two SCPDTs, calculated as $(A \cap B) \cup (B \cap C) \cup (C \cap A)$ (or shorthand $\cap 2$). This allows for identifying cases when a program has plagiarism-hiding source code modifications applied. However, this also has the potential for false positives. Hence, the term 'Broad' is used to describe this criterion. When both a score cut-off and consensus criteria are applied, it allows for a sub-set of similarity scores that consistently score highly to be identified for investigation.

The two consensus criteria have different purposes in what type of plagiarised program pairs are detected. The high confidence criterion is intended to identify the simplest verbatim or copy-and-paste cases of plagiarism. This is where there are few plagiarism-hiding modifications applied to the source code, and hence, the plagiarised programs are almost identical to their sources. This criterion will then indicate program pairs that are highly similar, and as such, have a high chance on being plagiarised. However, when plagiarism-hiding modifications are applied, it is expected that this criterion may miss plagiarised program pairs due to decreases in the measurement of similarity by the individual SCPDTs. To accommodate for this, the broad consensus criterion does not provide the same level of confidence of the high conference criterion that indicated program pairs are plagiarised. This is as any indicated pairs may be coincidentally similar, and therefore require further investigation before cases of plagiarism can be determined. In such cases, the individual similarity values need to be considered. For example, when the structure of the source code is pervasively transformed (e.g. by shuffling method declarations and statements), but the semantics and behaviour of the program remain relatively unchanged from its source. Such a case would be detected as having a high behavioural and semantic similarity, but mid or possibly low structural similarity [Cheers *et al.*(2020)].

The result of using this approach is that a subset of the program pairs selected using the applied cut-off will be indicated as being suspicious of plagiarism. Hence, by finding consensus amongst different SCPDTs, a semi-automatic method of indicating suspicious similarity scores is provided. This is an improvement over the current static cut-offs as it allows for suggesting out of a subset of scores, those that appear to be most suspicious without inaccuracies caused by limiting analysis to only a single tool. However, this approach is also expected to allow the ability to accurately indicate suspicious program pairs by selecting arbitrarily high cut-off values. This is as only a sub-set of similarity scores that are indicated as being suspicious, where those indicated must be consistently scored highly by each SCPDT. Hence, assuming each SCPDT is largely accurate on its own and the selected cut-offs are not exceedingly high, only a correctly suspicious sub-set of program pairs will be indicated. This will be demonstrated in the following case study, where the proposed approach is applied to detect known cases of plagiarism in undergraduate programming assignments.

Table 1.: Overview of evaluation data set characteristics.

Data Set	DS1	DS2
Submissions	84	76
Plagiarised	9 (11%)	5 (7%)
GitHub Sources	3	2

4. Case Study

The purpose of this case study is to demonstrate that the proposed approach is capable of accurately identifying assignment submissions that are suspicious of plagiarism by applying arbitrarily selected score cut-off values. In order to demonstrate the use of the proposed approach, it is applied to two data sets of undergraduate programming submissions, which both have known cases of plagiarism that were manually identified (with help from JPlag) and later confirmed by an impartial reviewer. This allows for providing a partial ground truth in this experiment, in that there are a fixed set of known and verified cases of plagiarism in the data set. Note, the term "partial ground truth" is used as there is the potential for cases of plagiarism that were unnoticed during the original assessment of the assignments.

The first data set (DS1) contains 84 assignment submissions, with 9 confirmed cases of plagiarism. The second data set (DS2) contains 76 assignment submissions, with 5 confirmed cases of plagiarism. None of the known cases of plagiarism are a result of students directly collaborating with peers in the same course offering. Instead, all of the known cases of plagiarism are a result of students basing their works on prior year assignments that were sourced from GitHub (note the GitHub sources were for prior year assignment specifications that were similar, but not identical). From DS1, it was found there were 7 submissions that were based on the same project found on GitHub, with the remaining two using other GitHub sources (i.e. a total of 3 GitHub sources of plagiarism). From DS2, it was found there were 4 submissions that were all based on the same GitHub source, with one more using a distinct source (i.e. a total of 2 GitHub sources of plagiarism). This is summarised in table 1.

The use of 'external' sources of plagiarism resulted in very interesting characteristics in terms of how the assignments were plagiarised, and how the indications of plagiarism are manifested in the similarity scores. In the simplest cases, the plagiarised assignments were almost verbatim copy-and-paste style plagiarism, with few modifications. However, in the more complex cases, the plagiarism was manifested as the paraphrasing of source code. This is where the plagiarised work is close to functionally identical to the GitHub source, but pervasively modified such that the programs look significantly different. When the data sets were analysed with JPlag, the submissions that were based on the same GitHub source would evaluate with low-to-mid levels of similarity (approximately 30% to 50% similar), ultimately causing doubt if or if not plagiarism was present. With human review, the submissions did look similar, however, without enough confidence to refer for academic misconduct on the grounds of collaboration. It was only after an extensive search of GitHub (using a custom-built web crawler that searched for keywords related to the course name and assignment specification), that the original source found, providing sufficient proof to make successful cases of plagiarism. By using these data sets, this case study will allow for evaluating the proposed approach in identifying known cases of plagiarism, that were

difficult to initially identify and subsequently prove.

4.1. Identification Process

The process to identify assignment submissions that are suspicious of plagiarism using the proposed approach is broken into four steps:

(1) Data collection.
(2) Similarity evaluation.
(3) Suspicious score analysis.
(4) Accuracy evaluation.

Firstly, the two data sets were collected from our institutions learning management system. As identified, the sources of plagiarism were not internal to the data sets, but hosted in GitHub in public repositories. Hence, in order to retrieve the known sources of the plagiarised works, a web crawler was used to search for GitHub repositories that included keywords related to the course code and assignment specification. For DS1, a total of 29 GitHub repositories matched the search criteria, while for DS2, a total of 111 GitHub repositories matched the search criteria. In both cases, the sources for the plagiarised assignment submissions were included in the matches, as well as many other unrelated repositories.

Secondly, the similarity scores of all assignment submissions compared to the repositories identified from GitHub are evaluated using all three tools. As the approach requires a structural, semantic and behavioural SCPDT, for the purpose of this case study, JPlag [Prechelt and Malpohl(2003)] is used as the structural tool, Graph ED [Cheers *et al.*(2021b)] is used as the semantic tool, and BPlag [Cheers *et al.*(2021a)] is used as the behavioural tool. This will provide three distinct perspectives of how each pair of submissions may be considered similar, that can be then used to find plagiarism from the consensus amongst the tools. Furthermore, it will also allow for identifying cases of plagiarism that may exist in the data set during the initial assessment and detection. Using these 3 SCPDTs, each is invoked comparing all of the collected assignment submissions with all collected GitHub repository pairs. This overall resulted in 7,308 similarity scores for DS1 ($84 \times 29 \times 3$), and 25,308 ($76 \times 111 \times 3$) similarity scores for DS2.

Thirdly, the similarity scores are analysed using the proposed approach. Initially three score cut-offs are applied as limiting criteria to the evaluated similarity scores. These three limiting criteria consist of the aforementioned similarity threshold of $x\%$, the top r scores, as well as a variation of top r referred to as the top $p\%$. The top $p\%$ is applied by taking the top $p\%$ of ordered similarity scores. The score cut-offs are applied with increasing x, r and p values to demonstrate the affect of using different arbitrarily selected configuration values. The two consensus criteria ($\cap 2$ and $\cap 3$) are then applied to each set of limited scores, with the resultant similarity scores each being considered to be indicative of potential plagiarism.

The applied score cut-off configuration values differ for each data set. For DS1, r values of 10, 15, 20, 25 are applied; p values of 10%, 15%, 20%, 25% are applied; and x values of 45%, 50%, 55%, 60% are applied. For DS2, r values of 5, 10, 15, 20, are applied; p values of 10%, 15%, 20%, 25% are applied; and x values of 40%, 45%, 50%, 55% are applied. The r values are selected based on the known number of cases of plagiarism is each data set. The closest multiple of 5 is selected as the first configuration value. Considerably greater r values then also applied to demonstrate that even with an arbitrarily selected large configuration value (compared to the known

number of cases) there is little to no decrease in accuracy. The p values are largely based on prior experience in academic marking and detecting cases of plagiarism. It is not unexpected that approximately 10% of assignment submissions are plagiarised, or have some other form of academic misconduct present in their development (as an upper bound); and is similar to the number of cases of plagiarism in the utilised data sets. The p value is then greatly increased to 25% to again demonstrate there is little to no decrease in accuracy by arbitrarily selecting a large score limit. The x values were selected arbitrarily from a low similarity score value.

Finally, the accuracy of the consensus-based approach for all configurations is evaluated. For the purpose of this case study, accuracy is reflected through the number of errors made when indicating plagiarised assignment submissions. An error is made when either the approach indicates a (presumably) innocent submission pair as plagiarised (i.e. FP or 'False Positive' error), or plagiarised similarity score as not plagiarised, (i.e. FN or 'False Negative' error), or TP (i.e. True Positive which are plagiarised similarity score is indeed plagiarised). The error count is calculated for each SCPDT as well as the proposed approach for each selected limiting configuration. This will allow for demonstrating the increased accuracy provided by the proposed approach for each arbitrarily selected limiting configuration.

4.2. Results

Tables 2, 3 and 4 display the results of this case study using the selected score limit values. For comparison purposes, the error counts of applying the score cut-offs to the results of each individual SCPDT are displayed, followed by the results of applying the consensus criteria. An interesting trend can be seen for both data sets when using the top r and top p% score limits. Initially at cut-offs close to the actual number of cases of plagiarism in each data set (9 for DS1, 5 for DS2), both the tools alone and the consensus-based approach report a noticeable (but not overly high) number of errors. However, as the cut-off values increase an interesting observation can be made. The error counts of the SCPDTs increase as expected, as more FP indications are being reported. However, for the proposed consensus based approach, the error counts tend to drop for $\cap 3$ and remain stable for $\cap 2$; but also change from being initial FN to FP errors. This is important as a FN indicates a student not being detected for plagiarism, while a FP indicates a presumably innocent student requires their work to be reviewed for indications of academic misconduct. Hence, when a consensus criterion is applied, a very low error count is evaluated even when the selected score limit is much greater than the known number of cases of plagiarism in each data set.

The decreasing error counts for the proposed approach is a result of the SCPDTs consistently evaluating the known plagiarised submissions with relatively high similarity scores. This results in the plagiarised submissions being above the score cut-offs, even when there are other presumably innocent submissions being reported by the tools. As the consensus-based approach requires two or three SCPDTs to indicate a submission as suspicious, this results in a much lower error count, especially when using relatively high score cut-offs. This implies that when using both consensus criteria, the most accurate indications of plagiarism can be found when using a much higher than expected score cut-off. And furthermore, the number of errors even with a high cut-off are similar to that of the number of errors made by the tools alone at the lowest recorded cut-offs. This further emphasises the benefit of using the consensus-based approach for identifying submissions that are suspicious of plagiarism.

Table 2.: Error counts of each tool with different limiting criteria r for DS1 and DS2. TP: True Positive, FP: False Positive, FN: False Negative, Error: FP+FN

$r =$	**Tool/Int.**	**TP**	**FP**	**FN**	**Error**
10	BPlag	7	3	2	5
	JPlag	8	2	1	3
	G. ED	6	4	3	7
	$\cap 2$	7	2	2	4
	$\cap 3$	6	0	3	3
15	BPlag	9	6	0	6
	JPlag	9	6	0	6
	G. ED	8	7	1	8
	$\cap 2$	9	3	0	3
	$\cap 3$	8	1	1	2
20	BPlag	9	11	0	11
	JPlag	9	11	0	11
	G. ED	9	11	0	11
	$\cap 2$	9	5	0	5
	$\cap 3$	9	1	0	1
25	BPlag	9	16	0	16
	JPlag	9	16	0	16
	G. ED	9	16	0	16
	$\cap 2$	9	5	0	5
	$\cap 3$	9	1	0	1

(a) Error Count for DS1

$r =$	**Tool/Int.**	**TP**	**FP**	**FN**	**Error**
5	BPlag	3	2	2	4
	JPlag	4	1	1	2
	G. ED	2	3	3	6
	$\cap 2$	3	0	2	2
	$\cap 3$	2	0	3	3
10	BPlag	4	6	1	7
	JPlag	5	5	0	5
	G. ED	3	7	2	9
	$\cap 2$	4	2	1	3
	$\cap 3$	3	0	2	2
15	BPlag	5	10	0	10
	JPlag	5	10	0	10
	G. ED	4	11	1	12
	$\cap 2$	5	3	0	3
	$\cap 3$	4	0	1	1
20	BPlag	5	15	0	15
	JPlag	5	15	0	15
	G. ED	4	16	1	17
	$\cap 2$	5	4	0	4
	$\cap 3$	4	0	1	1

(b) Error Count for DS2

Table 3.: Error counts of each tool with different limiting criteria p for DS1 and DS2. TP: True Positive, FP: False Positive, FN: False Negative, Error: FP+FN

$p =$	**Tool/Int.**	**TP**	**FP**	**FN**	**Error**
10% (r=9)	BPlag	6	3	3	6
	JPlag	8	1	1	2
	G. ED	6	3	3	6
	$\cap 2$	7	2	2	4
	$\cap 3$	5	0	4	4
15% (r=13)	BPlag	8	5	1	6
	JPlag	9	4	0	4
	G. ED	7	6	2	8
	$\cap 2$	8	3	1	4
	$\cap 3$	7	1	2	3
20% (r=18)	BPlag	9	9	0	9
	JPlag	9	9	0	9
	G. ED	9	9	0	9
	$\cap 2$	9	4	0	4
	$\cap 3$	9	1	0	1
25% (r=22)	BPlag	9	13	0	13
	JPlag	9	13	0	13
	G. ED	9	13	0	13
	$\cap 2$	9	5	0	5
	$\cap 3$	9	1	0	1

(a) Error Count for DS1

$p =$	**Tool/Int.**	**TP**	**FP**	**FN**	**Error**
10% (r=4)	BPlag	2	2	3	5
	JPlag	4	0	1	1
	G. ED	2	2	3	5
	$\cap 2$	2	0	3	3
	$\cap 3$	2	0	3	3
15% (r=9)	BPlag	4	5	1	6
	JPlag	5	4	0	4
	G. ED	3	6	2	8
	$\cap 2$	4	1	1	2
	$\cap 3$	3	0	2	2
20% (r=13)	BPlag	5	8	0	8
	JPlag	5	8	0	8
	G. ED	3	10	2	12
	$\cap 2$	5	3	0	3
	$\cap 3$	3	0	2	2
25% (r=18)	BPlag	5	13	0	13
	JPlag	5	13	0	13
	G. ED	4	14	1	15
	$\cap 2$	5	3	0	3
	$\cap 3$	4	0	1	1

(b) Error Count for DS2

Table 4.: Error counts of each tool with different limiting criteria x for DS1 and DS2. TP: True Positive, FP: False Positive, FN: False Negative, Error: FP+FN

$x =$	**Tool/Int.**	**TP**	**FP**	**FN**	**Error**
45%	BPlag	9	252	0	252
	JPlag	9	5	0	5
	G. ED	9	91	0	91
	$\cap 2$	9	12	0	12
	$\cap 3$	9	3	0	3
50%	BPlag	9	106	0	106
	JPlag	8	2	1	3
	G. ED	9	32	0	32
	$\cap 2$	9	6	0	6
	$\cap 3$	8	1	1	2
55%	BPlag	9	24	0	24
	JPlag	7	1	2	3
	G. ED	9	14	0	14
	$\cap 2$	9	3	0	3
	$\cap 3$	7	0	2	2
60%	BPlag	8	4	1	5
	JPlag	7	1	2	3
	G. ED	8	7	1	8
	$\cap 2$	9	2	0	2
	$\cap 3$	5	0	4	4

(a) Error Count for DS1

$x =$	**Tool/Int.**	**TP**	**FP**	**FN**	**Error**
40%	BPlag	5	1040	0	1040
	JPlag	5	3	0	3
	G. ED	5	2777	0	2777
	$\cap 2$	5	438	0	438
	$\cap 3$	5	2	0	2
45%	BPlag	5	733	0	733
	JPlag	5	1	0	1
	G. ED	5	1197	0	1197
	$\cap 2$	5	168	0	168
	$\cap 3$	5	0	0	0
50%	BPlag	5	362	0	362
	JPlag	4	0	1	1
	G. ED	5	371	0	371
	$\cap 2$	5	48	0	48
	$\cap 3$	4	0	1	1
55%	BPlag	5	139	0	139
	JPlag	3	0	2	2
	G. ED	5	99	0	99
	$\cap 2$	5	9	0	9
	$\cap 3$	3	0	2	2

(b) Error Count for DS2

The results of the $x\%$ threshold provide very different results compared to the top r or $p\%$ thresholds. This is caused by the number of indicated submissions being a function of the similarity scores, and not a fixed value. As the similarity score distribution evaluated by each tool differs, this results in a dramatic difference in the number of indicated submissions. As indicated by the results, BPlag and Graph ED have a much higher average similarity score, and hence, initially record many errors compared to JPlag when using the same $x\%$ threshold. However, as the threshold value increases, the error counts decrease quickly. Notably, while the $\cap 2$ consensus criteria reports many errors (due to the high average similarity of BPlag and Graph ED), the $\cap 3$ consensus criteria consistently reports a low error count. This indicates the $\cap 3$ consensus criteria has merit is use with a $x\%$ threshold score cut-off.

Notably when errors occur, the $\cap 2$ consensus criteria typically reports FP errors, while the $\cap 3$ method reports FN errors. Of the reported FP errors, many appear to be a result of missed indications of plagiarism. Apart from the very large FP error counts when using the $x\%$ score cut-off (that are simply a result of bad configuration values), many of the FP errors are a result of assignment submissions that have very similar designs as their presumed sources, and may in fact be a result of inappropriately using online resources to implement an assignment. Of the reported FN errors, this is typically where plagiarism-hiding source code modifications are applied to the plagiarised works. Hence, typically one tool will report a lower similarity score and hence not all tools will be in agreement that an indication of plagiarism is present.

Overall these results demonstrate the benefit of using the proposed consensus-based approach. By using the results of only a single SCPDT, one has to be lucky to determine the correct score cut-off. By using the proposed consensus-based approach of multiple SCPDTs, by selecting an arbitrarily high cut-off, the provided results indicate a remarkably high accuracy in indicating assignment submissions suspicious of plagiarism. Hence, with the proposed approach of combining the results of multiple SCPDTs and identifying SCPDT-based consensus, an accurate semi-automatic method of identifying assignment submissions that are suspicious of plagiarism is achieved.

5. Discussion and Conclusion

This work has proposed and evaluated a semi-automatic approach for indicating assignment submissions that are suspicious of plagiarism. This is through the described method of applying a score cut-off and finding consensus amongst the results of multiple SCPDTs. The selected SCPDTs each measure a distinct type of similarity, allowing for a multi-dimensional comparison of programs in terms of how they are similar.

From the results of the case study, the proposed approach appears to be less sensitive to selection of a score cut-off for identifying submissions with high similarity scores. Without using the consensus-based approach, typically the least errors would be found by using a cut-off that includes all plagiarised assignment submissions. However, this is of course difficult to select in a real world scenario. But by applying the consensus-based approach of combining the results of multiple SCPDTs, indications of plagiarism can be accurately found by using arbitrarily selected and often high score cut-off values. Hence, when using this approach, it can be suggested that a reasonably high cut-off value can be used with a satisfactory level of confidence. In terms of specific reusable cut-off criteria, arguably, the top $p\%$ cut-off is a more generic approach that can be reused on different data sets. This is as the score cut-off is not limited to a fixed number of submissions, and is determined by the data set size. In general, when using

this cut-off, while it may be expected to use a p value of 5-10 (roughly correlating to a reasonable upper bound of the number of assignments that may be plagiarised in a data set), the results of the case study indicate when using the proposed consensus criteria, accurate results can be found using a p value at 20-25.

Of the two utilised consensus criteria ($\cap 3$ and $\cap 2$), $\cap 3$ typically produced the least errors. This is a result of the criteria only indicating submissions that are evaluated with a high similarity score by all three tools, and hence are consistently suspicious of plagiarism. Hence it is referred to as the 'high confidence' consensus criteria. However, it was not free of either FP or FN errors. The $\cap 2$ criteria always reported some errors with all limiting criteria, however, this is a result of it being a 'broad' consensus criteria where it is intended to indicate any submission that is evaluated with a high similarity score by any two tools. It is not designed to be completely accurate, but to be less restrictive in order to accommodate for plagiarism-hiding transformations.

When applying either criteria results in errors, they are typically FP errors. This is a result of a program pair not part of the partial ground truth being indicated as suspicious. It is important to emphasise the importance of 'partial ground truth' here. Upon review of the FP results, some but not all of the indicated assignment submissions are worth investigation for potential academic misconduct. However, as no formal investigation of plagiarism has been undertaking for such submissions, they cannot be conclusively indicated as being a true FP or a TP that is missing from the ground truth. However, it must also be considered that the consensus-based approach cannot accommodate for the limitations and inherent inaccuracy of individual tools. Each tool provides its own independent indication of plagiarism through the evaluation of similarity. Hence, if one or more tools are inaccurate and miss reporting a high similarity score for a plagiarised assignment, an error will still occur due to it being missed via consensus.

As future work, it is planned to further this work in three directions. Firstly, it will be explored how to remove the dependency on applying a score cut-off to identify submissions with high similarity scores. This will be preliminary by applying clustering to the scores of each individual aspect to identify outlying high similarity scores, to ultimately provide an automatic approach for suggesting similarity scores that are suspicious of plagiarism. Secondly, further research into the meaning of different combinations of high similarity scores across the three aspects of similarity, i.e., what is the impact of an assignment pair having a high similarity across two aspects of similarity, but a low score in the remainder. This is largely to reduce FN errors, to ensure the proposed approach is accurate. And finally, to further evaluate the approach using diverse data sets from students of different year levels to empirically demonstrate its benefit.

References

[Ahadi and Mathieson(2019)] Ahadi, A. and Mathieson, L., 2019. A comparison of three popular source code similarity tools for detecting student plagiarism, *Proceedings of the Twenty-First Australasian Computing Education Conference*, ACE'19, 112–117.

[Ahtiainen *et al.*(2006)] Ahtiainen, A., Surakka, S., and Rahikainen, M., 2006. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises, *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea'06, 141–142.

[Allyson *et al.*(2019)] Allyson, F.B., Danilo, M.L., José, S.M., and Giovanni, B.C., 2019. Sherlock N-overlap: Invasive normalization and overlap coefficient for the similarity analysis between source code, *IEEE Transactions on Computers*, 68 (5), 740–751.

[Anjali *et al.*(2015)] Anjali, V., Swapna, T., and Jayaraman, B., 2015. Plagiarism detection for Java programs without source codes, *Procedia Computer Science*, 46, 749 – 758.

[Anzai and Watanobe(2019)] Anzai, K. and Watanobe, Y., 2019. Algorithm to determine extended edit distance between program codes, *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 180–186.

[Bertran *et al.*(2005)] Bertran, M., Babot, F.X., and Climent, A., 2005. An input/output semantics for distributed program equivalence reasoning, *Electronic Notes in Theoretical Computer Science*, 137 (1), 25 – 46, proceedings of the Fourth Spanish Conference on Programming and Computer Languages (PROLE 2004).

[Burrows *et al.*(2007)] Burrows, S., Tahaghoghi, S.M.M., and Zobel, J., 2007. Efficient plagiarism detection for large code repositories, *Software: Practice and Experience*, 37 (2), 151–175.

[Chae *et al.*(2013)] Chae, D.K., Ha, J., Kim, S.W., Kang, B., and Im, E.G., 2013. Software plagiarism detection: A graph-based approach, *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM'13, 1577–1580.

[Cheers *et al.*(2020)] Cheers, H., Lin, Y., and Smith, S.P., 2020. Detecting pervasive source code plagiarism through dynamic program behaviours, *Proceedings of the Twenty-Second Australasian Computing Education Conference*, ACE'20, 21–30.

[Cheers *et al.*(2021a)] Cheers, H., Lin, Y., and Smith, S.P., 2021a. Academic source code plagiarism detection by measuring program behavioral similarity, *IEEE Access*, 9, 50391–50412.

[Cheers *et al.*(2021b)] Cheers, H., Lin, Y., and Smith, S.P., 2021b. Evaluating the robustness of source code plagiarism detection tools to pervasive plagiarism-hiding modifications, *Empirical Software Engineering*, 26 (5), 1573–7616.

[Chen *et al.*(2010)] Chen, R., Hong, L., Chunyan Lü, C., and Deng, W., 2010. Author identification of software source code with program dependence graphs, *2010 IEEE 34th Annual Computer Software and Applications Conference Workshops*, 281–286.

[Cosma and Joy(2008)] Cosma, G. and Joy, M., 2008. Towards a definition of source-code plagiarism, *IEEE Transactions on Education*, 51 (2), 195–200.

[Cosma and Joy(2012)] Cosma, G. and Joy, M., 2012. An approach to source-code plagiarism detection and investigation using Latent Semantic Analysis, *IEEE Transactions on Computers*, 61 (3), 379–394.

[Curtis and Popal(2011)] Curtis, G. and Popal, R., 2011. An examination of factors related to plagiarism and a five-year follow-up of plagiarism at an Australian university, *International Journal for Educational Integrity*, 7 (1), 30–42.

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

[Evans(2020)] Evans, J., 2020. This university couldn't work out which students cheated, so it punished them all, *https://www.abc.net.au/news/2020-12-22/anu-computer-science-students-penalised-over-alleged-plagiarism/13004718*, accessed: 2021-05-01.

[Faidhi and Robinson(1987)] Faidhi, J. and Robinson, S., 1987. An empirical approach

for detecting program similarity and plagiarism within a university programming environment, *Computers & Education*, 11 (1), 11 – 19.

[Ferrante *et al.*(1987)] Ferrante, J., Ottenstein, K.J., and Warren, J.D., 1987. The program dependence graph and its use in optimization, *ACM Transactions Programming Languages and Systems*, 9 (3), 319–349.

[Fu *et al.*(2017)] Fu, D., Xu, Y., Yu, H., and Yang, B., 2017. WASTK: A weighted abstract syntax tree kernel method for source code plagiarism detection, *Scientific Programming*, 2017 (1), 103–126.

[Gitchell and Tran(1999)] Gitchell, D. and Tran, N., 1999. Sim: A utility for detecting similarity in computer programs, *SIGCSE Bull.*, 31 (1), 266–270.

[Grune and Huntjens(1989)] Grune, D. and Huntjens, M., 1989. Het detecteren van kopieën bij informatica-practica, *Informatie (in Dutch)*, 31 (11), 864–867.

[Jhi *et al.*(2011)] Jhi, Y., Wang, X., Jia, X., Zhu, S., Liu, P., and Wu, D., 2011. Value-based program characterization and its application to software plagiarism detection, *2011 33rd International Conference on Software Engineering (ICSE)*, 756–765.

[Joy and Luck(1999)] Joy, M. and Luck, M., 1999. Plagiarism in programming assignments, *IEEE Transactions on Education*, 42 (2), 129–133.

[Karp and Rabin(1987)] Karp, R.M. and Rabin, M.O., 1987. Efficient randomized pattern-matching algorithms, *IBM Journal of Research and Development*, 31 (2), 249–260.

[Lancaster and Codrin(2021)] Lancaster, T. and Codrin, C., 2021. Contract cheating by STEM students through a file sharing website: a Covid-19 pandemic perspective, *International Journal for Educational Integrity*, 17 (1), 1833–2595.

[Li *et al.*(2016)] Li, S., Xiao, X., Bassett, B., Xie, T., and Tillmann, N., 2016. Measuring code behavioral similarity for programming and software engineering education, *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, 501–510.

[Li and Zhong(2010)] Li, X. and Zhong, X.J., 2010. The source code plagiarism detection using AST, *2010 International Symposium on Intelligence Information Processing and Trusted Computing*, 406–408.

[Liu *et al.*(2006)Liu, Chen, Han, and Yu] Liu, C., Chen, C., Han, J., and Yu, P.S., 2006. GPLAG: Detection of software plagiarism by program dependence graph analysis, *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, 872–881.

[Luo *et al.*(2017)] Luo, L., Ming, J., Wu, D., Liu, P., and Zhu, S., 2017. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection, *IEEE Transactions on Software Engineering*, 43 (12), 1157–1177.

[Martins *et al.*(2014)] Martins, V.T., Fonte, D., Henriques, P.R., and da Cruz, D., 2014. Plagiarism Detection: A Tool Survey and Comparison, *3rd Symposium on Languages, Applications and Technologies*, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, *OpenAccess Series in Informatics (OASIcs)*, vol. 38, 143–158.

[Novak *et al.*(2019)] Novak, M., Joy, M., and Kermek, D., 2019. Source-code similarity detection and detection tools used in academia: A systematic review, *ACM Transactions Computing Education*, 19 (3).

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

[Pierce and Zilles(2017)] Pierce, J. and Zilles, C., 2017. Investigating student plagia-

rism patterns and correlations to grades, *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, 471–476.

[Pike(n.d.)] Pike, R., n.d. Sherlock plagiarism detector, *https://web.archive.org/web/20150323030146/http://rp-www.cs.usyd.edu.au/ scilect/sherlock/*, accessed: 2021-05-01.

[Prado *et al.*(2018)] Prado, B., Bispo, K., and Andrade, R., 2018. X9: An obfuscation resilient approach for source code plagiarism detection in virtual learning environments, *Proceedings of the 20th International Conference on Enterprise Information Systems - Volume 1: ICEIS* , 517-524, Funchal, Madeira, Portugal.

[Prechelt and Malpohl(2003)] Prechelt, L. and Malpohl, G., 2003. Finding plagiarisms among a set of programs with JPlag, *Journal of Universal Computer Science*, 8.

[Ragkhitwetsagul *et al.*(2016)] Ragkhitwetsagul, C., Krinke, J., and Clark, D., 2016. Similarity of source code in the presence of pervasive modifications, *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 117–126.

[Ragkhitwetsagul *et al.*(2018)] Ragkhitwetsagul, C., Krinke, J., and Clark, D., 2018. A comparison of code similarity analysers, *Empirical Software Engineering*, 23 (4), 2464–2519.

[Ramírez-de-la Cruz *et al.*(2014)] Ramírez-de-la Cruz, A., Ramírez-de-la Rosa, G., Sánchez-Sánchez, C., and Jiménez-Salazar, H., 2014. On the importance of lexicon, structure and style for identifying source code plagiarism, *Proceedings of the Forum for Information Retrieval Evaluation*, FIRE '14, 31–38.

[Rani and Singh(2018)] Rani, S. and Singh, J., 2018. Enhancing Levenshtein's edit distance algorithm for evaluating document similarity, *Computing, Analytics and Networks*, Singapore: Springer Singapore, 72–80.

[Roy *et al.*(2009)] Roy, C.K., Cordy, J.R., and Koschke, R., 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, *Science of Computer Programming*, 74 (7), 470 – 495.

[Schleimer *et al.*(2003)Schleimer, Wilkerson, and Aiken] Schleimer, S., Wilkerson, D.S., and Aiken, A., 2003. Winnowing: Local algorithms for document fingerprinting, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, 76–85.

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

[Yeo(2007)] Yeo, S., 2007. First-year university science and engineering students' understanding of plagiarism, *Higher Education Research & Development*, 26 (2), 199–216.

[Zhang *et al.*(2014)] Zhang, F., Wu, D., Liu, P., and Zhu, S., 2014. Program logic based software plagiarism detection, *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 66–77.

[Zhao *et al.*(2015)] Zhao, J., Xia, K., Fu, Y., and Cui, B., 2015. An AST-based code plagiarism detection algorithm, *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, 178–182.

[Zheng *et al.*(2018)] Zheng, M., Pan, X., and Lillis, D., 2018. CodEX: Source code plagiarism detection based on abstract syntax trees, *Proceedings of the 29th Irish Conference on Artificial Intelligence and Cognitive Science (AICS 2018)*, Dublin, Ireland.


Hayden Cheers received a Ph.D. in Software Engineering from the University of Newcastle, Australia in 2021. He currently splits his time between working as a research assistant at the University of Newcastle, Australia, and working as a senior software engineer. His research interests include source code similarity, plagiarism detection and applied software engineering.


Yuqing Lin received a B.Sc in Mathematics from Lanzhou University in China, and then received his Ph.D. in Computer Science from The University of Newcastle (Australia) in 2004. Now he is an Associate Professor at the University of Newcastle. His current research interests include discrete math, theoretical computer science and applied mathematics. Lin's interdisciplinary research explores how mathematics can be applied to real-world problems.


Weigen Yan received a Ph.D. in Mathematics from Xiamen University, China Australia in 2003. He is a professor in Jimei University in China. His research fields are combinatorics and graph theory, interested in enumeration of perfect matchings and spanning trees of graphs.