SOURCE CODE AND TEXT PLAGIARISM DETECTION STRATEGIES

Maeve Paris
School of Computing & Intelligent Systems
University of Ulster at Magee
m.paris@ulster.ac.uk
http://www.infm.ulst.ac.uk/~maeve

ABSTRACT

Plagiarism and collusion among students may be facilitated by the preponderance of material in electronic format and the ability to submit coursework online. A distinction has generally been drawn between plagiarism of text and plagiarism of source code, and different tools and metrics have been developed for either type. However, if a computer programming language is considered to be similar to a natural language (although it has a restricted syntax and vocabulary), computer-assisted text analysis techniques may be employed to assist the academic in detecting plagiarism in source code. So computational linguistics might inform software metrics, and vice versa.

Keywords

Plagiarism, Source code, Computer-assisted text analysis.

1. INTRODUCTION

Two distinct approaches have evolved to the process of automating the detection of plagiarism and collusion, depending on whether the material under investigation is a text document or a computer program. Surveys of software and services to assist detection (such as the publications from the Joint Information Services Committee (JISC)) generally focus on one or other of these approaches: plagiarism detection in text-based documents was examined in the *Technical Review of Plagiarism Detection Report* [1], which evaluated the performance of five products and services; plagiarism in computer programs was the focus of the report on *Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools*

4th Annual LTSN-ICS Conference, NUI Galway

[2], which tested two services. Neither report gave explicit consideration to this separation of concerns; there appears to be an implicit assumption that the practice of plagiarism differs according to whether the material is text or source code (even though the perpetrators, in this case, students, remain the same). Further studies compound this division: the LTSN Centre for Information and Computing Sciences [3] compared three source code analysis systems, highlighting mixed performance results, and further evaluations of text-based systems were carried out by the Learning Technologies Group [4]. Neither of the systems evaluated detected plagiarism although it was suggested the systems might help in the detection of collusion.

Despite this separation of concerns, computer programming languages have much in common with natural languages. They may operate with a more formal and restricted semantics and syntax, but they allow for a range of expression. It is possible to distinguish programming styles in code, in the same way as it is possible to distinguish writing styles in text. So, computer-assisted text analysis (CATA) techniques might be applied to the analysis of source code, and statistical methods from the domain of computational linguistics could be employed to enhance manual code inspection.

This study compares existing detection systems for text-based and source code plagiarism, and identifies the applicability of CATA techniques to source code analysis, illustrating this with a study of three approaches to the identification of plagiarism in a corpus of student assignments.

2. PLAGIARISM AND COLLUSION

The offences of plagiarism and collusion are closely related: both involve a failure to acknowledge sources, and collusion is a form of plagiarism. Most UK and Irish academic institutions have elaborated their own policy statements on these offences, and definitions vary in granularity and tone. Plagiarism covers a wide spectrum from extensive verbatim copying to phrase-level substitution. A useful indicator was offered from Lancaster University: plagiarism is “submission of work that is identical or substantially similar for assessment in more than one course […] passing off work as yours that is really the work of others (whether by other students or from other sources you have found)” [5].

Essentially, plagiarism relates to the submission of work for assessment which has been appropriated from another writer, but which fails to indicate this through the use of quotation marks or references. Collusion (or unauthorised collaboration) generally tends to be the result of students working together, but submitting work in such as way as to mislead an assessor with regards to the identification of the author.

3. EXISTING DETECTION SYSTEMS: TEXT

Systems to detect plagiarism in text are either standalone or accessed through a Web interface, and are largely based on text analysis techniques. Plagiarism or collusion in text can involve verbatim copying from a source document, or paraphrasing, and text analysis techniques can reveal inconsistencies in vocabulary, punctuation, spelling and word frequencies. Authoring styles can be identified through the calculation of statistics based on frequency counts such as the average number of words in a sentence, or sentences in a paragraph. Such statistics can also be generated through the use of CATA tools for concordancing, and a Key Word In Context (KWIC) index can be exploited to reveal parallels in aligned texts.

Online services (evaluated in the JISC report [1]) are available to detect plagiarism in text: plagiarism.org creates a representation of each document submitted to its database which it can check against other representations in the database as well as documents on the World Wide Web; standalone products include CopyCatch and WordCHECK. WordCHECK maintains an internal database of documents against which it checks submitted documents by matching keyword profiles. CopyCatch measures pairwise similarity between texts based on word frequencies.

4. EXISTING DETECTION SYSTEMS: SOURCE CODE

Modern detection systems tend to adopt a metrics-driven approach in order to produce a measurement of the closeness or similarity between two programs. These metrics can be drawn from the domain of software engineering, including such measurements as the cyclomatic complexity of the control flow of a program, or the number of each type of data structure; or they may be drawn from the domain of linguistics, including the choice of variable names, the use of certain layout conventions such as indentations, and the quantity and quality of comments. Taken as a whole, such measurements can assist in building a profile of a programming or authoring style [6]. Most detection systems are based on this lexical-structural approach, where programs are tokenised, and profiles are created and subsequently compared with other program profiles in order to quantify closeness.

The selection of items to include in a profile depends on how plagiarism in source code is quantified. Jones [7] offered a useful definition of plagiarism detection, characterising it as a problem of pattern analysis, based on plagiarising transformations which have been applied to a source file. Such transformations include “verbatim copying, changing comments, changing white space and formatting, renaming identifiers, reordering code blocks, reordering statements within code blocks, changing the order of operands/operators in expressions, changing data types, adding redundant statements or variables, replacing control structures with equivalent structures” [7]. So these patterns are a combination of software and linguistic elements:

Some academic institutions have developed their own internal detection systems (for example, Big Brother [8] at the University of Glasgow, which has been used to identify instances of collusion in order to improve assessment procedures), but there are also online services available through a Web interface. The most prominent of these services are sim (Software Similarity Tester), YAP (Yet Another Plague), MOSS (Measure of Software Similarity) and JPLAG.

Little is known about the implementation details of most of these services, presumably because if this information was in the public domain, it could be employed to evade plagiarism detection. Sim tokenizes source code and compares strings using pattern-matching algorithms based on work from the human-genome project [9]; and YAP also tokenizes source code, but retains only those tokens which are concerned with the structure of the program [10].

MOSS is a free Internet service which can be applied to C, C++, Java, Pascal, Ada, ML, Lisp, or Scheme programs. Batches of programs are submitted to the MOSS server, which returns HTML pages with lists of pairs of programs with similar code, and highlighted passages in individual programs. The service is based on the syntax of a program, rather than the algorithms which drive the program [11]. JPLAG (another free service) compares submitted programs in pairs and supports Java, C, C++ and Scheme files. Unlike other services, JPLAG also supports natural language text, even though it claims to have inferior performance than that expected on source code. Like MOSS, JPLAG focuses on structure and syntax, and is based on the assumption that while plagiarists may vary the names of variables and classes, they are less likely to change a program’s control structure. A technical report provides details the detection algorithm [12].

The JISC report [2] provided performance evaluations on MOSS and JPLAG, and concluded that there was little consensus between the services in identifying plagiarism. JPLAG was considered easier to use, but it supported fewer programming languages than MOSS, and crucially, it could not handle programs which do not parse. The report concluded that the results returned were widely different. Another comparison report produced by the LTSN [3] also revealed patchy performance and inconsistencies, and a commentator [13] identified a major problem with tools such as JPLAG which rely on control structure metrics to detect pairwise similarities: "these primitive constructs – the IF, THEN and ELSE statements – are used in about the same ratio in just about every program" [14]; as a consequence, tools which rely on such metrics may generate false positive results.

5. CATA AND PROGRAMMING LANGUAGES

Reports point to inconsistent performance by available source code plagiarism detection systems [2,3], and most contemporary systems base metrics largely on the structure and syntax of the programs under investigation. The results of such an approach may be of little use to instructors: perhaps CATA techniques might be of greater use. Some commentators have identified this, although from the opposite perspective. Clough [14], for example, identified similarities between methods used for source code and text plagiarism detection, including "replacement of synonyms, re-ordering of sentences, insertion and deletion of text, change of author style". He concluded that "methods used for software plagiarism detection may well work for text also". If that is the case, then the reverse may also be true: if there are similarities between detection of plagiarism in both areas, then the methods used for text plagiarism detection may work for software plagiarism detection also.

Programming languages typically consist of a lexicon (vocabulary), and syntax (structure), which determines whether a program is capable of being compiled or parsed. Even though these may be restricted in nature, they can still be treated as a language from a linguistic perspective: natural languages also have a lexicon (the words in a vocabulary), and syntax (the structure of phrases and sentences), which governs whether strings of words are grammatical (well-formed) or ungrammatical (ill-formed). Although the richness of the lexicon and syntax is a function of the individual programming language, the individual programmer still has a certain amount of flexibility in achieving the desired functionality of source code, including the choice of implementation algorithm, the nomenclature of classes and variables, and the physical layout of code on the screen through the user of whitespace and indentation, for example. Control flow statements, variable names and selection of expressions all allow for the individual programmer to exercise some degree of choice; this in turn enables the identification of authoring styles: "the stylistic influence of an individual on algorithm implementation within the constraints of a given programming language is limited but can be identified to some extent as traits or tendencies in the expression of logic constructs, data structure definition, variable and constant names and calls to fixed and temporary data sets" [15]. Other commentators have made similar observations: "it is possible to identify the author of a section of program code in a similar way that linguistic evidence can be used for categorizing the authors of free text" [16].

Similarity between programs can also be determined through examination of spelling and grammar, and tools developed for computer-assisted text analysis should be able to assist in quantifying such data: "many programmers have difficulty writing correct prose. Misspelled variable names […] and words inside comments may be quite telling if the misspelling is consistent. Likewise, small grammatical mistakes inside comments or print statements, such as misuse or overuse of em-dashes and semicolons might provide a small additional point of similarity between two programs" [17].

6. ANALYSIS

The aim of the analysis was to ascertain the extent to which a concordance program could assist an experienced instructor in the detection of source code plagiarism or collusion. The use of the concordance tool was also compared with a manual inspection of the same files, and the results obtained from the JPLAG service.

A corpus of Java source code programs was obtained from a group of first year BSc Honours Computer Science students, submitted for assessment based on the development of a static method which converted a percentage value to a letter grade. The program should prompt for a name and percentage, use the method to determine the grade and output the summary onscreen[1]. The corpus comprised thirty-five files, two (StudentGrade.java and GradeWork.java) were similar, apart from a difference in class name and substitution of the word 'got' for 'achieved' in the output, and in addition, Prac7.java and Percentage.java had an identical (unusual) static method. The remaining files in the corpus were not substantially similar, and some of the files would not compile successfully.

[1] Heather Sayers, University of Ulster.

6.1 Manual Inspection and JPLAG

An assessor examined the thirty-five files in printed form: the very similar files were identified within fifteen minutes, and the remaining similarities were identified after a further five minutes. The assessor felt this method was impractical for large collections of files, and she commented that she was specifically looking for similar variable or class names, similar loops, programs with comments removed or misspelled, and unusual naming schemes.

The files were then submitted to the JPLAG server using an Applet, and results were returned to a webpage (Figure 1).

Matches:

prac7Q2.java	->	prac7.java (82.3%)	question4.java (61.5%)	ReadMarks.java (23.0%)
MarksJG.java	->	ConvertMarks.java (51.5%)	ExampleQ7.java (35.9%)	ReadMarks.java (27.7%)
prac7.java	->	question4.java (46.1%)	ReadMarks.java (25.6%)	
ExampleQ7.java	->	ConvertMarks.java (43.4%)		
GradeAssign.java	->	GradesPT.java (22.5%)	GradesMS.java (15.2%)	

Figure 1: JPLAG summary results

Some of the files were rejected (JPLAG only handles files which can be parsed), so twenty-six of the files were considered and matches identified in eleven of them. The results were somewhat different from the manual inspection and the top match identified (82.3% similarity) was between prac7Q2.java and prac7.java (see Figure 2).

The assessor was not satisfied that these files showed evidence of plagiarism, since all that they shared was the control structure, but the way the control structure was used was substantially different in each case. JPLAG did not identify the clear cases of plagiarism, as the programs concerned did not parse. So, the main disadvantages to the JPLAG approach are that it is largely based on control structures, and it cannot handle programs which cannot be compiled

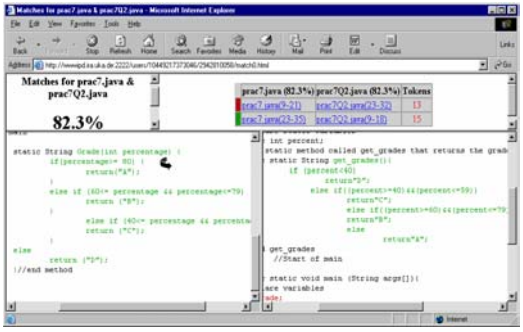

Figure 2: JPLAG detailed feedback

6.2 Concordance

The thirty-five files were input to the Concordance tool [19]. Although this tool is designed to work with text, it can be modified to handle source code files. Adjustments need to be made in terms of specifying syntax and lexicon items: a stop list needs to take into account the most common words which are found in Java programs (public, static, void, main, and so on). The tool batch converted java files to text files while retaining file and line information.

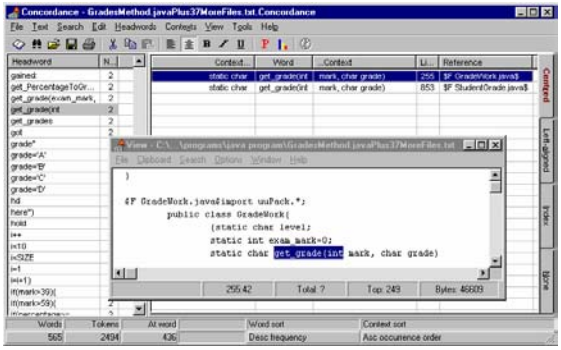

Figure 3: Concordance results window

Once the concordance was generated, the results were displayed sorted in descending frequency. By selecting suspect headwords with a high occurrence frequency, the assessor could see these keywords in context on the right, and by clicking on the line of code, the program itself appeared in a separate window with the line highlighted. The files are referenced, and the assessor located similarities between StudentGrade.java and GradeWork.java within five minutes, despite having little experience of a concordance application. The assessor felt that the application enabled her to locate instances of possible similarity more quickly and accurately than through manual inspection or JPLAG, and it also facilitated quick identification of files and lines of code. Drawbacks included the need to scroll through the list of headwords to make judgements on which suspect occurrences should be investigated further, and also the configuration of the application which had to be set up to handle computer files through specific stop lists for different programming languages.

7. Conclusions

Software and services to detect plagiarism and collusion have evolved separately, depending on whether the suspect material is text-based or source code; however, it might be more useful to combine software metrics with computational linguistics when dealing with source code. Although programming languages have a more formal and restricted lexicon and syntax than natural language, it is still possible to distinguish a programming style, and this small study indicates that techniques from

computer-assisted text analysis might usefully be applied to the detection of source code plagiarism. Existing systems concentrate largely on control structures, which may not be so important given that many student assignments are relatively short: control structures tend to be repeated, and in practice, plagiarists tend to modify variable or class names. Programs which cannot be compiled as just as likely to be plagiarised, and most current systems cannot handle these. In the longer term, tools might be developed which combine metric-based profiles with concordancing features. It is important to remember that tools cannot prove that plagiarism has occurred; they merely indicate its possibility, and it is up to the instructor to decide if the offence has taken place.

8. REFERENCES

[1] Bull J., Collins C., Coughlin E., Sharp. D *Technical Review of Plagiarism Detection Report* JISC (2001).

[2] Culwin F., MacLeod A., Lancaster T. *Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools* JISC, (2001).

[3] LTSN, *Tools to assist detecting plagiarism* Available at: http://www.dcs.warwick.ac.uk/ltsn-ics/resources/plagiarism/tools/comparison_performance.html (2002).

[4] Lee S. *Plagiarism Detection Study* http://www.oucs.ox.ac.uk/ltg/reports/plag3.htm (2001).

[5] Lancaster University, *Undergraduate Guide to Essay Marking and Essay Quality*, Number 6 http://www.comp.lancs.ac.uk/sociology/ugessay.html (2000).

[6] Gray A., Sallis P., MacDonell S. Software Forensics: Extending Authorship Analysis Techniques to Computer Programs. *Proceedings of the 3rd Biannual Conference of the International Association of Forensic Linguists* pp 1 – 8 (1997).

[7] Jones E., Plagiarism monitoring and detection – towards an open discussion. *Proceedings of 7th annual CCSC Central Plains Conference,* Branson, Missouri, April 6-7, (2000).

[8] Irving R., Plagiarism Detection: Experiences and Issues. *JISC Fifth Information Strategies Conference, Focus on Access and Security*, London, (2000).

[9] Gitchell D., Tran N., Sim: a utility for detecting similarity in computer programs. *Proceedings of the thirtieth SIGCSE technical symposium on computer science education* New Orleans, Louisiana, pp 266-270 **(**1999).

[10] Wise M YAP3: improved detection of similarities in computer programs and other texts. *SIGCSE'96*, Philadelphia, Feb 16 – 17, (1996).

[11] MOSS http://www.cs.berkeley.edu/~aiken/moss.html (2002).

[12] Prechelt L, Malpohl G, Philippsen M *JPlag: Finding plagiarisms among a set of programs*, Universität Karlsruhe, Institut für Programmstrukturen und Datenorganisation, (2000).

[13] Stutz M., Catching Computer Science Cheaters. *Wired News* http://www.wired.com/news/technology/0,1282,10464,00.html (1998).

[14] Clough P., *Plagiarism in natural and programming languages: an overview of current tools and technologies*. University of Sheffield. http://www.dcs.shef.ac.uk/~cloughie/plagiarism/HTML_Version/ (2000).

[15] Sallis, P, Aakjaer A, MacDonell Software Forensics: old methods for a new science. *Proceedings of Software Engineering: Education and Practice* IEEE Computer Society Press, pp 481-485 (1996).

[16] DeVel O, Anderson A, Corney M, Mohay GM Mining Email Content for Author Identification Forensics. *SIGMOD RECORD* 30(4): 55-64(2001).

[17] Weeber S.A., Spafford E.H., Software forensics: can we track code to its authors? *Computers & Security* 12(6): 585-595, (1993).

[18] Watt RJC *Concordance*. http://www.rjcw.freeserve.co.uk/ (2002).