Automatic Plagiarism Detection

C. DALY and J. M. HORGAN
School of Computer Applications
Dublin City University

Abstract: Existing systems for detecting plagiarism in computing are limited in that, while they are reasonably successful in identifying pairs or groups who submit similar work, they do not indicate who the original author is. The system RoboProf which we present here is a learning environment for Java programming, with a novel technique to detect plagiarism which overcomes this limitation. For this paper, we used RoboProf to monitor a class of 283 first year students of computing. Our results show that nearly 40% of the class plagiarised one or more pieces of course work, and that those who plagiarised had significantly higher failure rates, and significantly lower average marks, in the end-of-semester examination than those who did all their course work from scratch. The failure rates increased and the average marks decreased with the amount of plagiarism.
Keywords: Plagiarism; programming; watermark.

INTRODUCTION

Research from surveys on plagiarism indicates that it is prevalent in university courses worldwide. A report from the Office of Research of the Department of Education of the United States [1] points to studies which reveal that cheating ranged from 9% to 95% of the student bodies at American institutions of higher learning. A survey of university students in Moscow found that most students cheated some time or other in their student life [2]. Recent British surveys suggest that over half the students sampled were involved in a range of cheating behaviour [3].

Over the last decade there has been an explosion of electronic detection systems for detecting plagiarism in computing. The many suggested techniques include attribute counting [4], use of standard software metrics [5] and examination of redundant code [6]. There is even a server on the web, the Moss server at Berkeley [7], which will process a list of programs and indicate which ones are suspiciously similar. All of these use pairwise comparisons of students' work, and they are reasonably successful in pointing to pairs or groups who submit similar work. They are limited however in that they do not indicate who is the original author of the work. In addition, these methods are unlikely to work on short segments of code.

RoboProf, the system which we present here, is a learning environment which automatically assesses programming exercises, and has a built-in facility for detecting plagiarism which overcomes these limitations. This technique is a development of the observation of Plauger [8], that a 'fingerprint' in the form of invisible white space is often left inadvertently in the source code by the programmer, and that this could be used to prove theft. Subsequently Brassil [9] and Berghel [10] showed how to use this fingerprint to protect copyright. What RoboProf does is to actually insert a watermark into each program, invisible to the author, but recognisable to the computer if the same program is submitted again. With this watermark, it is possible to identify the original author of the work and subsequent plagiarists. Our system has the additional property that it is able to detect plagiarism on small segments of code. It works, without the need to store previous records, if students use work from a previous year, and it works even if the student modifies the program extensively, as long as they do not inadvertently change the watermark.

We proceed in the next section to an overview of RoboProf. In §3 we describe how RoboProf detects plagiarism, determining the incidence and extent of plagiarism in a large class of first year students of computing. In §4, we compare the performance in the end-of-semester examination for the plagiarists and those who do their work from scratch. The conclusions are discussed in the final section.

ROBOPROF

RoboProf is a learning environment for Java programming which generates and assesses programming exercises automatically. In operation, RoboProf runs as a Java Servlet [11] in conjunction with a Web server producing web pages for each student. Each page is generated dynamically, based on how the student is performing, and RoboProf can generate problems based on student ability.

RoboProf loosely follows the progression rules of the old style text-adventure games [12], where one had to solve a problem in order to increase the amount of space available for exploration. In practice RoboProf presents a series of simple programming exercises, which necessitate writing a short section of code. It uses an applet to compile and test the program on the student's machine. The reason for using the student's own machine for doing this is to reduce the load on the server (compiling and running Java Programs require a lot of CPU cycles). Students can use any Java-enabled web browser to access RoboProf at any time, and to have their progress relative to other members of the class monitored and conveyed to them. They can request to see the expected output alongside the output of their own program for each set of input data, in order to help to improve their understanding. These procedures are documented in Daly [13].

For this research, RoboProf is used by a first year class taking an introductory course in Java. Throughout the semester, students are required to complete a series of simple programming exercises, automatically generated by RoboProf. The purpose of these is to get students writing programs as early as possible, and to develop their low-level programming skills by frequent practice at writing short programs. They can resubmit their program as often as they wish, without penalty, until they are satisfied with the standard they have achieved. Each week, RoboProf analyses the submissions, logs the results and returns them to students. RoboProf is meant to provide formative assessment; the purpose is to give the student as much practice as is necessary to ensure that they have confidence in writing short programs, and have mastered the basic syntax and semantics of the language before moving on to longer programs, where design issues and problem solving are important.

IDENTIFYING PLAGIARISM

In addition to generating and assessing programming exercises, RoboProf contains a technique which detects plagiarism. It is designed so that it has 'FILE-READ' and 'FILE-WRITE' permissions on the student's computer. When a student submits a program to RoboProf, the compiling and testing applet first stores the program on the server, then modifies the program by adding an identifying watermark unknown to the student. This watermark is a binary code comprising the student's ID, the assignment number, the academic year and a checksum. Spaces and tab characters at the end of a line are used to form the binary code. This white space is added to the end of main method, which is unlikely to be changed if the program is modified. Most text editors do not show excess white space at the end of a line, so that the code is not visible to the students. Each time a program is submitted, the student's ID number is compared with the version in the original watermark, hence identifiing the plagiarists. RoboProf presents a plagiarism report, via the web, in which every detected case of plagiarism is highlighted.

This method of detecting plagiarism has many advantages over the traditional pairwise comparison methods:

- it identifies the author of the program;
- it will detect plagiarism even if the assignment had been copied from a student's work of a pre-

vious year. This is possible because the year the original work is done is encoded into the watermark;

- it can detect plagiarism in short sections of code;
- it is programming language independent. Although RoboProf is designed for Java programming, this technique could be used in any programming learning environment;
- it works even when the program has been extensively modified, as long as the watermark is undisturbed;
- plagiarism is detected as soon as the program is submitted. It is not necessary to collect a lot of pairs and run an analysis program.

For this course, a total of 51 programming exercises were generated by RoboProf for completion by the end of the semester. Figure 1 charts the incidence of plagiarism detected for the 283 students who participated. It was found that a total of 102 (35.7%) plagiarised at least one programming exercise. It can be seen from Figure 1 that the number of programs plagiarised was as high as 19 for some students.

A limitation of this method is that it cannot detect retyped versions of another's program; it works only if the student submits an electronic copy of the program. Hence, the amount of plagiarism detected by RoboProf would be an underestimate of the true amount.

PLAGIARISM AND PROGRAMMING PERFORMANCE

At the end of the semester, there was a three hour end-of semester invigilated examination where plagiarism is impossible. The examination, consisting of five programs, each testing different programming techniques, is presented and corrected by RoboProf.

To examine the difference in performance in the examination for the varying levels of plagiarism, we divided the students into four groups with increasing numbers of plagiarised exercises, as given in Figure 2;

Figure 1: Incidence of Plagiarism

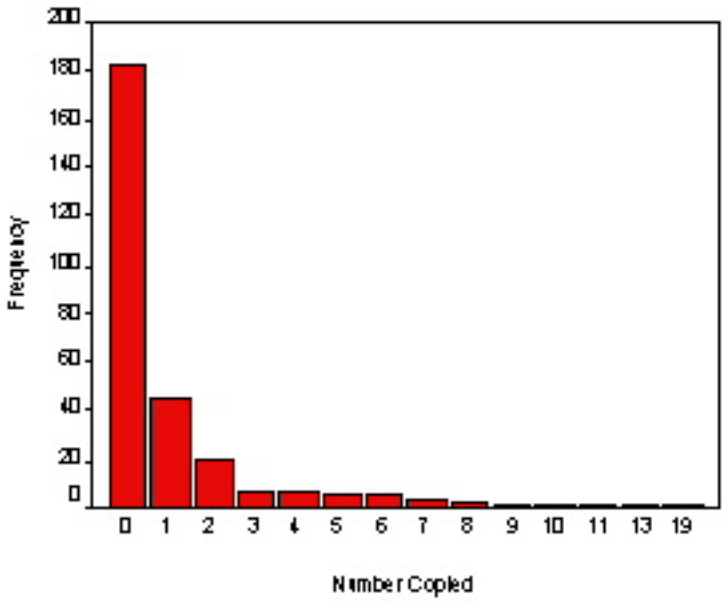


Figure 2: Plagiarist Groups

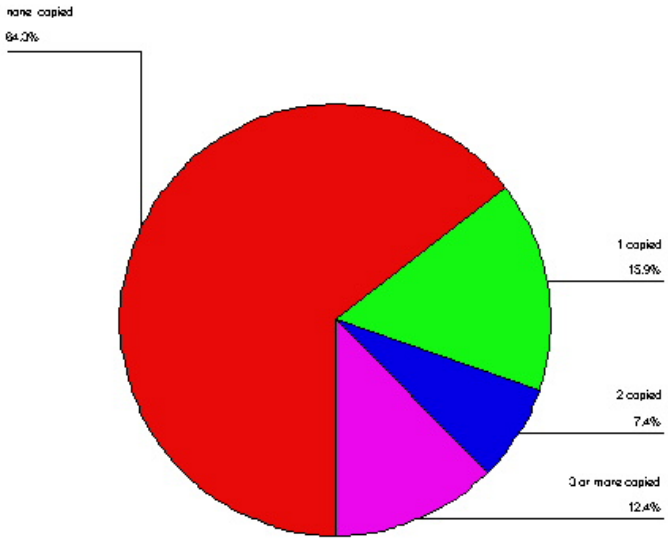


Table 1: Dunnett's Test of Comparison of the Plagiarists and Non-Plagiarists

Number Copied	N	Mean	Diff. from Control	SE of Diff.	Sig.
0 (Control)	182	53.215			
1	45	42.116	-11.010	5.038	.041
2	21	36.001	-17.206	6.974	.021
3	35	29.891	-23.324	5.585	.000

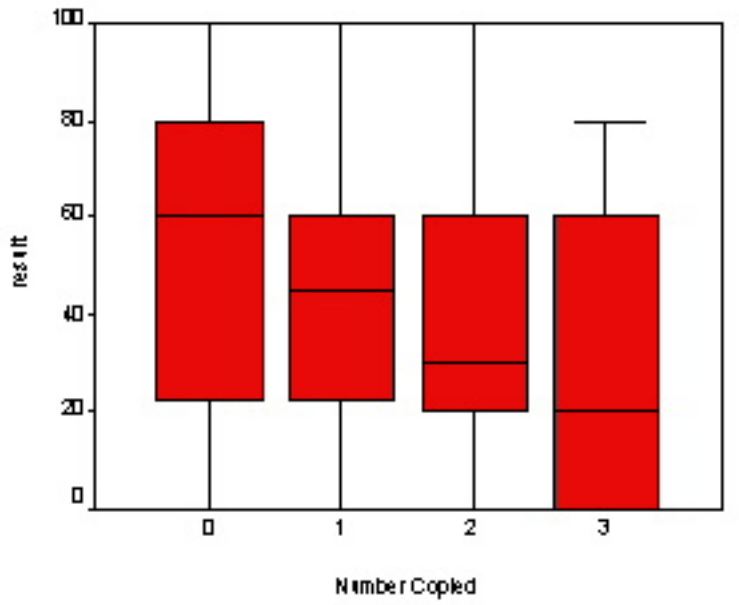
those who did all their who work (64.3%), those who copied one (15.9%), two (7.4%) and three or more (12.4%).

Average Marks

A one-way analysis of variance, carried out to test the difference in performance in the end-of-year examination between the groups, yielded an F-statistic of 7.594 which is significant with $p < .001$. Dunnett's multiple comparison test [14] was invoked to compare each of the plagiarist groups with the non-plagiarist control group with respect to performance in the end-of-year examination. The results, given in Table 1, show that all three plagiarist groups differ significantly from the non-plagiarist group with respect to the result achieved in the end-of semester examination; the difference increases with the rate of plagiarism; the average performance of those who plagiarised three or more is just under 30% while those who did their own work achieved an average of over 50%. Boxplots of the results for each of the four groups, given in Figure 3, confirms the difference in performance of the groups; 50% of the non-plagiarist group obtained a mark of 60% or above, while no more than 25% of the other three groups obtained a mark in this range. Not surprisingly, the group who plagiarised three or more had the greatest incidence of low marks; 50% of them obtained a mark less than 20% compared to 25% of the group that did not copy.

Failure rates

Figure 3: Distribution of Examination Results



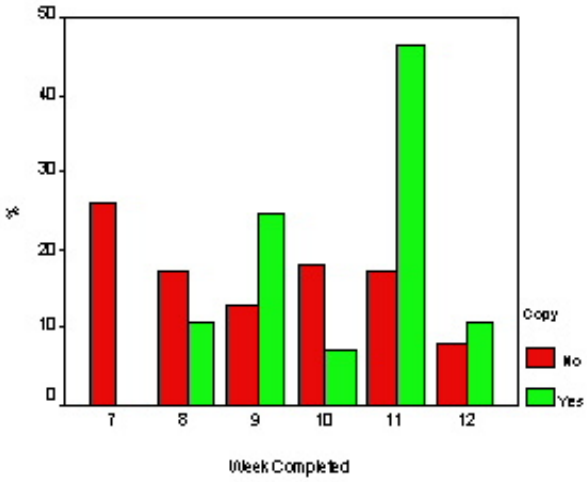
To pass the examination it was necessary to obtain at least 40%; Table 2 gives the pass/fail rates in each of the four groups. A chi-squared value of 14.383 indicates that the failure rates are significantly different ($p < .002$) in the different groups. The failure rate increases with the extent of plagiarism; those who completed all the work from scratch had a failure rate of 28%, while nearly 60% of those who plagiarised three or more laboratory tests failed.

Table 2: Pass/Fail Rates

	Number of Programs Plagiarised				
	0	1	2	3+	Overall
Pass	131(72.0%)	28(62.2%)	10(46.6%)	15(42.9%)	184(65.0%)
Fail	51(28.0%)	17(37.8%)	11(52.4%)	20(57.1%)	99(35.0%)

Completion Patterns.

Figure 4: Solve Time



Our final analysis examines the times taken to complete the RoboProf exercises. 104 students completed the full set; of these 28 plagiarised and 76 did their work from scratch. The completion times, given in Figure 4, show that the non-plagiarist group tended to complete the full set earlier than those who plagiarised; over 25% of the non-plagiarists finished in the seventh week at least a week ahead of the plagiarists; the first set completed by the plagiarised did not arrive until the eighth week. Nearly 50% of those who plagiarised did not complete the full set until the eleventh week, suggesting that these may have relied on the early solvers.

SUMMARY AND DISCUSSION

Effective learning of computer programming necessitates a lot of hands-on experience in writing simple programs. With large classes, it is almost impossible to monitor and correct these manually. Another problem with large classes is that students who lag behind on practical work may be tempted to plagiarise, banking on the class size reducing the risk of being discovered and thus defeating the whole point of the exercises.

RoboProf overcomes both of these problems. In the first place, by assessing the laboratory work automatically, it is possible to give a large amount of practical work without an increase in work for the lecturer. Secondly, RoboProf has a facility of detecting plagiarism which, if used at the first signs of its occurrence, may reduce it at the later stages.

For this research we monitored 283 first year students taking a one-semester course in Java. We attempted to discourage plagiarism, at the outset, by placing a policy on the module web site, which pointed out that the sharing of code was not acceptable and that the penalties were severe. We even informed the students that we had software to detect plagiarism. Despite this, we found that nearly 40% of the class plagiarised the practical work to some degree or other, and over 10% of the students could be deemed serious plagiarists (copying three or more exercises).

The plagiarists did significantly less well in the end-of-semester examination than their honest peers; their failure rate was significantly greater and their average mark was significantly lower. It was possible to monitor their performance on the practical work throughout the semester, and we found that the plagiarists lagged behind the other students in terms of when they completed. Clearly, the student who copied relied on the early solver.

References

[1] S. Maramark, and M. B. Maline (eds), *Academic Dishonesty among College Students. Issues in Education*, (Office of Educational Research and Improvement, Washington, DC, 1993).

[2] Y. Poltorak, Charting behaviour among students of four Moscow universities *Higher Education*, 1995, 225-246.

[3] http://www.le.ac.uk/tlu/tanrep1.html

[4] H. Jankowitz, Detecting Plagiarism in Computer Science Programs . *The Computer Journal*, 31(1), 1998, 1-8.

[5] G. Whale, Identification of program similarity in large populations, *The Computer Journal*, 33(2), 1990,

[6] J. Traxler, Cheating in Pascal programming assessments with large classes, *Proc. 3rd Annual Conf. Teach. Comp.*, 1995, 340-355,

[7] Moss server at Berkeley, information at URL http://www.coe.berkeley.edu/EPA/EngNews/98S/EN1S/aiken.html

[8] P. J. Plauger, Fingerprints, *Embedded Systems Programming*, June 1994.

[9] J. Brassil,et al, Electronic marking and identification techniques to discourage document copying, *Technical Report*, ATT Bell Laboratories.

[10] H. Berghel, Watermarking cyberspace, *Comm. ACM*, 40(1), 1997, 19-24.

[11] J. Hunter, *Java Servlet Programming*, (O'Reilly Associates, 1998).

[12] Colossal Cave Adventure documented on the web at URL http://people.delphi.com/rickadams/adventure/.

[13] C. Daly, RoboProf and an introductory programming course , *Proc. 4th Annual Conf. Innov. Tech. Comp. Sci. Ed. ITiCSE 99* Cracow, Poland, 1999.

[14] E. R. Dougherty, *Probability and Statistics for Engineering, Computing and Physical Sciences*, (Prentice-Hall, 1990).