A PLAGIARISM DETECTION SYSTEM

John L. Donaldson, Ann-Marie Lancaster and Paula H. Sposato

Computer Science Department
Bowling Green State University
Bowling Green, Ohio 43403

ABSTRACT

The problem of plagiarism in programming assignments by students in computer science courses has caused considerable concern among both faculty and students. There are a number of methods which instructors use in an effort to control the plagiarism problem. In this paper we describe a plagiarism detection system which was recently implemented in our department. This system is being used to detect similarities in student programs.

INTRODUCTION

In most computer science courses, the completion of programming assignments is part of the course requirements. In many cases, these assignments contribute significantly to a student's grade; thus, each student is usually expected to work independently. However, it is unlikely that there is any method for preventing students from collaborating with one another in the process of writing their programs.

The problem of students handing in programs which are not their own has caused concern among both faculty and students. It is the feeling of many faculty members that incidents of plagiarism are quite prevalent. Recently, the computer science department at Carnegie-Mellon University formed a committee to deal with the problem of plagiarism as it applies to programming assignments. Surveying several computer science departments, this committee found that the plagiarism problem is of widespread concern [1].

There are a number of methods that instructors have used in an effort to control the plagiarism problem. Some instructors inform the students that they may be asked to answer questions about their programs. A student whose performance on exams does not correlate with his or her scores on programming assignments is a likely choice for such a question-answer period. However, for most instructors, this is an extremely unpleasant and time-consuming task.

Most instructors rely upon the grader of the assignments to detect occurrences of plagiarism. In our department, this works well for upper level courses. The number of students is generally between 25 and 30. The instructor for the course usually grades the assignments, rather than having it done by a graduate assistant. In addition, the assignments are somewhat complex, so that the occurrence of similar assignments is more apt to be noticed. In contrast, relying upon the grader appears to be inadequate to detect plagiarism among assignments in the lower level classes. Graduate assistants usually grade these assignments and, while the size of each of these classes may not be large, each assistant is usually grading for several sections of a course having similar assignments. It has been the experience in our department that graduate students rarely detect occurrences of plagiarism.

Another approach to the problem of detecting plagiarism in programming assignments is that of employing an automatic detection system. One such system is described in [2]. This system assigns to each program a four-tuple (n1, n2, N1, N2):

- n1 - the number of unique operators;
- n2 - the number of unique operands;
- N1 - the total number of occurrences of operators;
- N2 - the total number of occurrences of operands.

The four-tuples of programs are compared and two programs having the same four-tuple are candidates for further investigation. The author of this method argues that the probability of two different assignments having identical four-tuples is slight. However, for programs of the size typical of introductory level classes, this is not always the case.

Another plagiarism-detecting system has been incorporated in a program analysis system called Instructional Tool for Program Advising (ITPAD) [3]. This system builds a graph to represent the structure of each student's program. It then compares pairs of programs by counting attributes of this representation.

In this paper, we describe an automatic detection system which has recently been implemented at the computer science department at Bowling Green State University. Our goal in developing this system was to enable it to detect similarities between the structures of two programs. Students in introductory programming courses who copy usually use rather simplistic methods for disguising their plagiarizing. These commonly used methods do not often change the structure of the program. Some of these methods are:

1) renaming variables;
2) transposing statements when the ordering of the statements does not effect the result;
3) altering format statements;
4) breaking up single statements such as declarations and output statements.

Our detection system was designed so that programs which differ only in one or more of the four ways listed above would, in most cases, be tagged as similar.

By picking out similar programs, this detection system indicates likely cases of plagiarism. It can then be used by the instructor as a tool to determine which programs warrant further examination.

In the following section, we describe the algorithms used in our detection system. The system is designed to accept the source code version of each student assignment from a file. For each assignment, it creates a table containing information pertaining to the structure and content of the assignment. After all assignments have been processed, the system compares the table of each assignment with the table of every other assignment in the file. The results of these comparisons, as well as the information in each table, is printed for the instructor's use. Currently, the system is designed for programs written in FORTRAN, COBOL or BASIC. However, in this paper we will discuss only the analysis of FORTRAN programs.

DESCRIPTION OF DETECTION SYSTEM

The detection system has been implemented using the SNOBOL4 programming language. The analysis of the assignments is done in two phases, a data collection phase and a data analysis phase. In the data collection phase, each assignment is read, line by line, and information is gathered on the characteristics of the assignment. In the data analysis phase, these characteristics are compared and tables of results are built.

Data Collection Phase

Assignments are characterized in two ways. First, the number of times certain types of statements occur is computed. Separate counters are established for each statement type. As each statement in an assignment is processed, the appropriate counter is incremented. When the detection program has finished scanning an entire assignment, the counter values are saved in a two-dimensional array to be used in the second phase. For FORTRAN assignments, the detection system keeps track of the following:

1. Total number of variables
2. Total number of subprograms
3. Total number of input statements
4. Total number of conditional statements
5. Total number of loop statements
6. Total number of assignment statements
7. Total number of calls to subprograms
8. Total number of statements of type 2-7

Secondly, assignments are characterized by the order in which statements occur in an assignment. Certain statement types were chosen to be significant in describing the structure of an assignment. Each of these types was assigned a single code character. As the statements in an assignment are processed, a string of code characters is built; the left-to-right order of the code characters in the string corresponds to the top-to-bottom order of the statements in the assignment. When the detection program has completed scanning an assignment, the string is saved as one element of a one-dimensional array. When execution of the data collection phase has been completed, this array will contain all the strings for all assignments in the file.

The code characters and the FORTRAN statements to which they are assigned are:

V	- Declaration statement
S	- Subroutine or function definition
C	- Call or execute statement
R	- READ statement
I	- IF (conditional expression) THEN DO
X	- logical IF
H	- WHILE (conditional expression) DO
D	- DO loop
E	- END IF, END WHILE, or CONTINUE
=	- Assignment statement

A sample assignment and output

A sample FORTRAN assignment is shown in figure 1. The table of counters and statement order sequence generated by the data collection phase from this assignment are shown in figure 2. Note that the number of statements does not include line numbers 1, 9, 10, 37, and 39 through 47. The number of conditional statements includes line numbers 16, 22, 28, and 31. Note also that the assignments on line numbers 11, 13, 15, and 27 are not counted. The coded statement order sequence does not show the location of ELSE DO, WRITE, and FORMAT statements.

```
          STUDENT NAME
          INTEGER KEY(10),ANSWR(10),IDNUMS(50),SCORE(50)
          INTEGER COUNT,RIGHT,BEST,IDCODE,I,N
          CHARACTER*1 GRADE
          COUNT = 0
          RIGHT = 0
          BEST = 0
          N = 10
          WRITE(6,60)
          WRITE(6,61)
          READ(5,50) (KEY(I),I=1,N)
          WHILE (.TRUE.) DO
             READ(5,51,END=900) IDCODE,(ANSWR(I),I=1,N)
             COUNT = COUNT + 1
             DO 10 I=1,N
                IF (KEY(I) .EQ. ANSWR(I)) THEN DO
                   RIGHT = RIGHT + 1
                END IF
    10       CONTINUE
             SCORE(COUNT) = RIGHT
             IDNUMS(COUNT) = IDCODE
             IF (BEST .LT. RIGHT) THEN DO
                BEST = RIGHT
             END IF
             RIGHT = 0
          END WHILE
    900   DO 11 I=1,COUNT
             IF (SCORE(I) .GE. BEST-1) THEN DO
                GRADE = 'A'
             ELSE DO
                IF (SCORE(I) .GE. BEST-1) THEN DO
                   GRADE = 'C'
                ELSE DO
                   GRADE = 'F'
                END IF
             END IF
             WRITE(6,62) IDNUMS(I),SCORE(I),GRADE
    11    CONTINUE
          WRITE(6,63)
    50    FORMAT(10(I1))
    51    FORMAT(I4,10(I1))
    60    FORMAT('1STUDENT')
    61    FORMAT(' NUMBER      SCORE      GRADE')
    62    FORMAT('0 ',I4,8X,I2,8X,A1)
    63    FORMAT('1 ')
          STOP
          END
```

FIGURE 1
Sample student FORTRAN assignment

```
STUDENT NAME

     NO. OF STATEMENTS:                                   34
     NO. OF VARIABLES AND ARRAYS:                         11
     NO. OF SUBPROGRAMS:                                   0
     NO. OF INPUT STATEMENTS:                              2
     NO. OF CONDITIONAL  STATEMENTS:                       4
     NO. OF LOOPS:                                         3
     NO. OF ASSIGNMENT STATEMENTS:                        13
     NO. OF CALL AND EXECUTE STATEMENTS:                   0

     CODED STATEMENT ORDER SEQUENCE:
        VVV====RHR=DI=EE==I=E=EDI=I===EEE
```

FIGURE 2
Sample output from the data collection phase
(see figure 1)

Data Analysis Phase

The second function of the detection program, data analysis, is accomplished in two phases. In the first, a measure of the degree of similarity or difference between the counters of any two assignments is calculated. The result is a single integer comparator for each pair of assignments. In the second phase of the analysis process, the statement order sequences of each pair of assignments are compared.

Phase I

Three different methods were implemented to determine a similarity or difference factor.

Sum of the differences. The first of the counter algorithms is called the Sum of the Differences. To determine the extent of the difference between two assignments, corresponding counter values are subtracted, and the absolute values of the differences are summed.

Count of similarity. The second of the counter algorithms measures the degree of similarity between the counters of each pair of assignments. Each similarity factor is initially zero. Then, corresponding counter values are compared; if they are equal, the factor is incremented by one. If they are not equal, the factor value remains unchanged.

Weighted count of similarity. This method is an extension of the above algorithm. Instead of adding one when corresponding counter values of two assignments are equal, the corresponding weight is added to the similarity factor for the pair of assignments under consideration. The weights used are specified by the instructor.

Phase II

The statement order sequences of each pair of assignments are compared to determine if the structures of the assignments are similar enough to indicate plagiarism. In the discussion that follows, the statement order sequences being compared will be referred to as X and Y.

The algorithm begins by compressing X and Y such that identical characters, appearing in succession, are reduced to a single character. Thus, the string 'VVVR===HI==EDI==EEE' would be reduced to 'VR=HI=EDI=E'. Then, the compressed forms of X and Y are compared, character by character. As soon as a mismatch occurs, the process is halted and the match is considered unsuccessful. If the end of either string is reached and no mismatch has occurred, the message:

'***SUCCESSFUL MATCH***' is printed.

DISCUSSION OF ALGORITHMS

The purpose of the detection system is to simplify the process of comparing the student assignments. It would be cumbersome for the instructor to attempt a comparison of the counters and character strings by hand. It has been our goal to develop algorithms whose results would indicate which pairs of assignments were quite similar and should be examined by the instructor. By considering both content and structure, we hope to detect at least those plagiarism cases which employ those methods most commonly used for disguising copying and collaboration.

Counter Algorithms

Three separate algorithms were implemented for comparing the counter values for each assignment. The sum of the differences algorithm gives the instructor some indication of how two assignments differ in content. If the sum is small, the implication is that the contents of the assignments are similar to some degree. However, the values generated by this algorithm for a pair of assignments should be analyzed by the instructor relative to the values for all other pairs of assignments in the file. It is assumed that all assignments being compared are meant to solve the same programming problem. Therefore, small differences do not necessarily mean that two assignments were plagiarized. However, if the difference between two assignments is very small, while at the same time the differences for the majority of the class are large, then perhaps these two assignments deserve closer scrutiny.

In contrast to the above algorithm, the counter of similarity indicated how similar the contents are of two assignments. It shows the instructor how many of the items counted are exactly equal. It does not show, however, which of these items are the same. This comparator must also be analyzed in relation to all similarity counts for the class. Notice that the maximum value this count can have for any two FORTRAN assignments is eight. Thus, if the assignments are simple, it is not unexpected that the counts of similarity would be close to eight.

The weighted count of similarity has one major advantage over the sum of differences and the count of similarity. It enables the instructor to weight each of the statement types relative to the demands of the particular programming problem assigned. For example, if the assignment is relatively simple, the instructor might expect there to be no subprograms. Counting this item would, in this case, be misleading, since all assignments will be alike with respect to this one item. Assigning a value of zero to the weight for the count of subprograms would effectively eliminate the item from consideration. Again, the values resulting from this method, as with the two previous counts, should always be evaluated relative to the entire class.

String Comparison Method

The string comparison algorithm will isolate those assignments which have the same order of statements, that is, the same basic structure. Since students may try to hide their copying by breaking up single statements into multiple statements, this algorithm considers a succession of the same statement type as one statement.

In a language with structured programming constructs, the string method accurately reflects the program structure and permits simple analysis. Inclusion of END statements in the string insures that if the strings match, then the structure of nesting levels must also match. The use of the GOTO statement would make it much easier to disguise a program's structure; however, at Bowling Green, students are restricted from using the GOTO. We would not recommend applying this method to programs making heavy use of the GOTO.

CONCLUSION

It is certainly safe to say that neither the detection system described in this paper nor any other detection system will find all occurrences of plagiarism. There is an inherent tradeoff between a highly discriminatory system, which overlooks some instances of cheating, and a less discriminatory one which flags many dissimilar programs. However, the experience gained from continued use of our system will permit further refinement of our methods of analysis. Already, the system has proved to be a useful tool. It has enabled instructors who have used it to detect cases of plagiarism that had gone unnoticed by the graders. The deterrent effect of catching even just a few cases of cheating should help to curb this troubling problem.

REFERENCES

[1] Shaw, M., Jones, A., Kneuven, P., McDermott, J., Miller, P., and Notkin, D., "Cheating Policy in a Computer Science Department," ACM SIGCSE Bulletin 12, 2 (July, 1980), 72-76.

[2] Ottenstein, K.J., "An Algorithmic Approach to the Detection and Prevention of Plagiarism", CSD-TR 200, Purdue University (August, 1976).

[3] Robinson, S. and Soffa, M., "An Instructional Aid for Student Programs", ACM SIGCSE Bulletin 12, 1 (February, 1980), 118-127.