AC 2008-2554: ENTROPY BASED VERIFICATION OF ACADEMIC INTEGRITY

Thomas Doyle, McMaster University

Adrian Ieta, Murray State University

Sheng Qian, McMaster University

Entropy Based Verification of Academic Integrity

Abstract

The usage of online submission provides an efficient means of facilitating course components; especially those with large enrolment. However, this convenience is not without caveat as student solutions are then as easily distributed as they submitted for credit. While services exist to combat academic dishonesty, such as online comparison databases, privacy concerns have been raised about their usage. We have developed and implemented an entropy based method for the verification of academic integrity. This tool was implemented primarily for a freshman undergraduate programming course with a yearly enrolment of more than 1000 students. Even with significant resources and effort dedicated to ensuring academic integrity, the result was difficult to quantify. Further compounding the challenge was the fact that computer program source code has rigidly defined constructs and syntax, thus a simple text comparison could indicate a high level of similarity that might implying a lower level of integrity. Conversely, two logically identical programs could be written with different variable names where a simple text comparison could indicate a higher level of integrity. Rather than performing a straight comparison, our entropy based method generates a symbolic library of the file and then analyses the library structure against all other file libraries as a measure of academic integrity; this method defeats the short comings of the aforementioned methods. This paper will present our entropy based method and its high level of success verifying the academic integrity of large sets of assignment submissions.

1. Introduction

Assessment of student work is one of the few methods we have as instructors to communicate if the student has attained an acceptable mastery of the subject. As instructors we also use this "conversation" to gauge our own success at the transfer of incremental pieces of the pedagogical puzzle. This feedback is used to refine and optimize the course for the student to get the most from the experience. This optimization is heavily based upon the premise that students have responded to the best of their ability using their own work.

When this premise is found to fail, a common response by the instructor is to reduce the course weight of the component(s) where academic integrity may be questioned (e.g. computer programming laboratories). While this appears to limit the gains of the dishonest student, it will also demotivate the honest students by:

1) Placing less emphasis on core aspects of the course building blocks, and
2) Continuing to reward questionable methods with high marks.

In the effort to limit the effects that cheating will have on the overall course assessment (macro-view), the instructor may actually be encouraging the cheating because from the student perspective the cheaters continue to be rewarded on an incremental basis (micro-view); this is regardless of how little the component may be worth.

Considerable time and resources go into refining courses and assessing the student work. At first, reducing the weight of the problem course component(s) appears to be an efficient method of handling the problem. However, the result effects all students and their learning experience.

To address the root of the problem, verification of academic integrity is required. While analysis of each assignment by the instructor is ideal, for large courses it is impractical. Once the verification task requires several people, the detection becomes more difficult and less efficient. In addition, the burden of proof lies with the instructor which requires the assembly of documentation and (in many cases) the filing of the academic dishonesty charge with the department or University. The processing of a single academic dishonesty charge can take an inordinate amount of time that redirects significant resources from the operation, development, and improvement of the course.

In an effort to efficiently handle academic integrity verification, services such as TurnItIn.com (http://www.turnitin.com/) have become a key tool in assessing written assignments. However, there have been both academic and legal[1,2] objections to such services, making the usage of such external services questionable.

Academic integrity validation for computer source code poses a slightly different challenge because the syntax and structure of the logic are rigidly defined. With the use of some modern integrated-development-environments (IDE) even the individual's own programming style is reformatted to that of the IDE's designers. A number of tools exist for the detection of programming plagiarism[3-7], two popular online tools are Measure Of Software Similarity[8] (MOSS - http://theory.stanford.edu/~aiken/moss/) and Shared Information Distance[9] (SID - http://genome.math.uwaterloo.ca/SID/).

Both MOSS and SID offer online analysis based upon the same principle of information content comparison. The method presented in this paper uses a measure of information content (entropy) by comparing the size of symbolic libraries generated by a commonly available compression method.

What we offer is an open and efficient method of locally comparing student work, a tool to simplify the verification of academic integrity, and a quantitative measure for the processing of academic dishonesty cases.

2. Background

The first year engineering programme at McMaster University has common curricula. The academic year is divided into three terms and the Engineering Computation course runs in all three (Fall, Winter, Summer). The Fall and Winter terms each have an enrolment of 450 students and the Summer term normally has 100 students. Students are not required to have any experience programming prior to entering the programme, but by the end of first year all students are expected to be proficient in writing their only software solutions.

The operation of the course requires fifty to sixty undergraduate teaching assistants to run 10 laboratory and 10 tutorial section per week (10 laboratories and 10 tutorials per term). While

assignments had individualized components (e.g. based upon student number) the task of verifying academic integrity by hand was significant. Detection across lab and tutorial sections was difficult and time intensive. Processing such cases was equally as difficult and generally reserved for the most severe.

Our initial approach to solving the problem of academic integrity verification was to investigate the generation of source code signature waveforms for comparison (similar in concept to Schleimer [9]) or to require the use of our own editor (similar to Vamplew's work[3]) with embedded authorship and checksum information. However, the entropy based measure of information content proved to have an elegant and efficient implementation with very positive results; both academically and pedagogically.

2.1 Information and Entropy

As Shannon said, "the semantic aspects of communication are irrelevant to the engineering aspects" [10]. In the measure of information it is not so important what was communicated, but rather what technically could have been communicated. For example, in a simple binary transmission the choice of what can be transmitted (technically) is 0 or 1; regardless of the meaning of transmitting a 0 or 1. The measure of the information (X) in the communication is given by the logarithm of the number of communication choices or symbols (p).

$$X = log_b p \tag{1}$$

The base (b) of the logarithm is somewhat arbitrary, but with a binary communication the base of 2 is generally used. For our example, a single bit is transmitted and either option is equally possible. The amount of information communicated here is 1 bit and for any communication with 2^N choices that are each equally probable, the amount of information communicated is N-bits.

However, there is the inconvenience that communication should have meaning. Structured communication, the ordered selection of symbols, cannot have all possible symbols being equally probable. Consider the English alphabet: the letter 'T' appears much more often than the letter 'Z' in common communication. Now consider that the probability of the letter 'T' being followed by the letter 'Z' is statistically much lower than being followed by an 'H'. The measure of information when we consider the statistical probability of each symbol in communication is called the *entropy (H)*. Measuring the information when considering the probability, the expression for n independent symbols, each with probability p is written as,

$$H = -\ [p_1\ log_b p_1 + \ p_2\ log_b p_2 + \cdots + \ p_n\ log_b p_n, \text{ or} \tag{2}$$

$$H = -\ \sum_{i=1}^{n} p_i\ log_b p_i. \tag{3}$$

2.2 Lossless Compression

Compression has two classifications: lossless and lossy. As the name suggests, "lossless" compression will produce an identical copy of the original data when uncompressed. For example, when compressing a paper for a conference, you prefer its data returned to the original, rather than approximated. "Lossy" compression produces an approximation of the original data for reduced storage; however, the original data is lost. Lossy JPEG compression on photograph data is a common example.

Entropy bounds the theoretical limit on lossless compression; to go beyond this limit would require some loss of information, thus no longer "lossless". The ideal compression algorithm would provide an accurate measure of information.

While no lossless compression method can provide a perfect calculation of entropy, some do provide an excellent approximation. The high level of compression is in part achieved by the compression application dynamically generating its own symbol library and probability statistics. The result is that data is no longer stored discretely, but instead as a library of symbols; repeated sequences of symbols are stored as their own symbol, etc. For our purposes, the gzip compression method was used.

3. Academic Integrity Verification

The goal of our application is to quickly identify problem submissions from large data sets and provide objective, qualitative data.

3.1 Method

Our application employs the algorithm illustrated in figure 1. Files are preprocessed

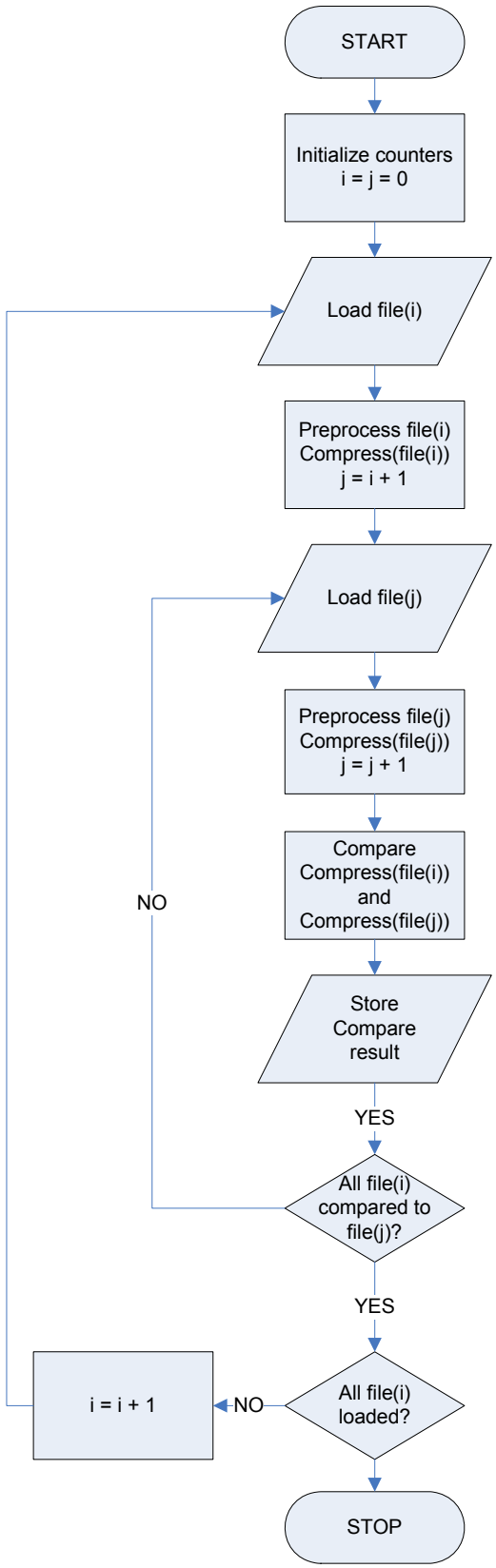

Figure 1: Academic Integrity Verification Method

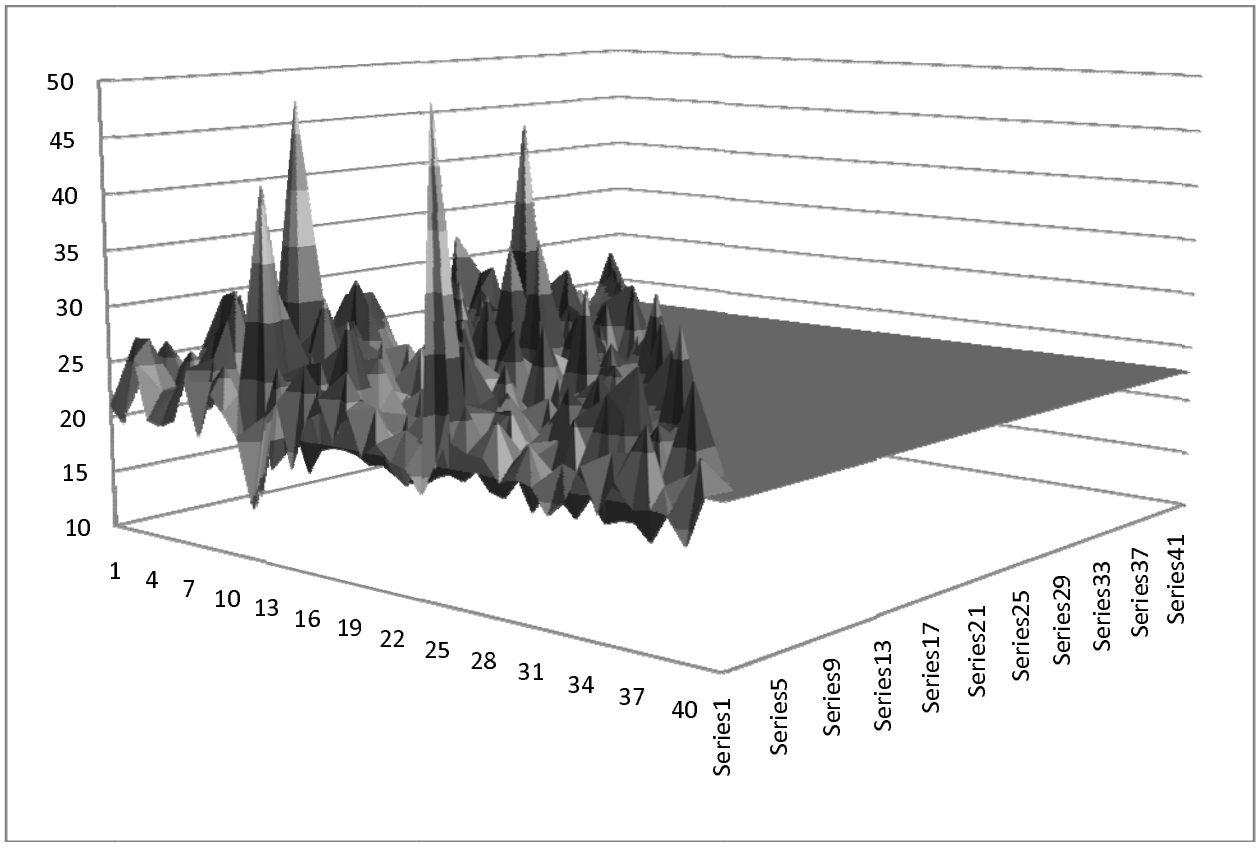

Figure 2: Comparison of 41 student submissions against average similarity.

```
double FuncAverage(double Sum, int Count);  // prototype which computes a
double stdv(int Count, double Sum, double sum2); //prototype which comput


        int main(void)
        {
        FILE *input;                            // input file variable
        FILE *output;                           // output file variable

        int Input_Status;                        // stores status value retu
        int Count = 0;                           // stores number of integer
        double NumberRead;                       // stores number read from
        double Sum = 0  ;                        // stores sum of integers
        double sum2 = 0;                         // stores sum of integers s
        double Average;                          // stores average of intege
        double standardevation;          // stores the standrad deviation
        double largestnumber = -10000.0;
        double Smallestnumber = 10000.0;
        int Number_of_dig_abv_mean = 0 ;
//-------------------------------------------------------------------------
// welcome massage
        printf("Hello and welcome to the program.\n\n");


// preparing 2 files for reading and writing

        input = fopen ("lab3input.txt", "r");
        output = fopen("lab3output.txt", "w");

//read in data from text file

        for (Input_Status = fscanf(input, "%lf", &NumberRead);
                 Input_Status != EOF;
                 Input_Status = fscanf(input, "%lf", &NumberRead))
                {
                Count++;                                 // counting number
                Sum += NumberRead;                // as each number is read,
                sum2 = sum2 + (NumberRead * NumberRead);
                Average = FuncAverage(Sum, Count);      // call to function
```

```
double FuncAv(double Sum1, int Count);  // prototype which computes averag
double stdv(int Count, double Sum1, double sum2); //prototype which compu

        int main(void)
        {
        FILE *input;                            // input file variable
        FILE *output;                           // output file variable

        int Input_Status;                        // variable declaration
        int Count = 0;                           // variable declaration
        double Read;                      // variable declaration
        double Sum1 = 0  ;                       // variable declaration
        double sum2 = 0;                         // variable declaration
        double Average;                          // variable declaration
        double standardevation;          // variable declaration

// first message viewed by user
        printf("Hello and welcome to the program.\n\n");


// Files for reading and writing

        input = fopen ("lab3input.txt", "r");


//this will read in the file

        for (Input_Status = fscanf(input, "%lf", &Read);
                 Input_Status != EOF;
                 Input_Status = fscanf(input, "%lf", &Read))
                {
                Count++;                                 // total amount of
                Sum1 += Read;              // the number keeps getting added t
                sum2 = sum2 + (Read * Read);
                Average = FuncAv(Sum1, Count);  // function which uses two
                standardevation = stdv(Count, Sum1, sum2);      // function

//statement printed to the screen
                printf("Number Read =  %8.3lf    Sum1 = %8.3lf    Average =
```
Figure 3: The top valued comparison for similarity using our entropy based comparison method.

to remove commenting and the results are sorted in descending order.

3.2 Results

The results from our academic integrity verification application are graphed in figure 2 and the top valued comparison displayed in figure 3.

Figure 2 displays the comparison values of 41 student submissions. The graph contrasts the average value of similarity (green plane) against all possible combinations of the 41 student submissions. This is an introductory programming course where sections of code may be supplied for student use, thus the average similarity cannot be expected to be the same from assignment to assignment. In addition to the academic integrity verification algorithm, the application will also sort which assignments to review based absolute number, percentage, or standard deviation. We have also implemented a grouping algorithm based on student number.

Figure 3 illustrates one of the core problems presented; a straight text comparison would not identify these two submission samples as duplicates. A cursory visual inspection clearly shows the code is logically identical with one submission attempting to appear different using several different variable names.

4. Discussion and Conclusion

By using the approximation of entropy from the gzip compression algorithm we have developed an academic integrity verification system that may be employed privately by the instructor to analyse a large set of student source code submissions. The dynamic creation of a symbol library permits the indirect comparison of the symbols and overcomes the shortcomings of straight text comparison.

The academic integrity verification application is an excellent method of detecting problem submissions and it provides an objective measure of similarity relative to all other submissions. Instructors interested in applying this tool may download it from our website at http://www.cybranetics.com.

5. Future Work

Our goal is to further develop our series of tools [11-13] that may assist academics in assessment and data management. Future work on this application will make the user interface more intuitive and automate the graphical result.

Bibliography

1. Foster, A., “Plagiarism-Detection Tool Creates Legal Quandary”, The Chronicle of Higher Education, May 17, 2002. < http://chronicle.com/free/v48/i36/36a03701.htm >
2. Canadian Broadcast Corporation, “McGill student wins fight over anti-cheating website”, CBC News, January 16, 2004. < http://www.cbc.ca/canada/story/2004/01/16/mcgill_turnitin030116.html >
3. Mozgovoy, M., “Desktop Tools for Offline Plagiarism Detection in Computer Programs”, Informatics in Education, v5-1, pp. 97—111, 2006.
4. Vamplew, P., Dermoudy, J., “An Anti-Plagiarism Editor for Software Development Courses”, Proceedings of the 7th Australasian conference on Computing Education - Volume 42, Newcastle, New South Wales, Australia, pp. 83 – 90, 2005.
5. Brin, S., Davis, J., García-Molina, H., “Copy Detection Mechanisms for Digital Documents”, Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pp. 398-409, San Jose, California, USA, 1995.
6. Joy, M., Luck, M., “Plagiarism in Programming Assignments”, IEEE Transactions on Education, v42-2, pp. 129 – 133, 1999.
7. Parker, A., “Computer Algorithms for Plagiarism Detection”, IEEE Transactions on Education, v32-2, pp. 94 – 99, 1989.
8. Chen, X., Francia, B., Li, M., McKinnon, B., Seker, A., “Shared Information and Program Plagiarism Detection”, IEEE Transactions on Information Theory, v50-7, pp. 1545-1550, 2004.
9. Schleimer, S., Wilkerson, D. S., Aiken, A., “Winnowing: Local Algorithms for Document Fingerprinting”, Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, pp. 76 – 85, 2003.
10. Shannon, C., Weaver, W., “The Mathematical Theory of Communication”, University of Illinois Press, 1949.
11. Ieta, A., Doyle, T. E., Kucerovsky, Z., and Greason, W. D. "Challenges and Options Related to Scaling Raw Scores in Engineering Education," The International Network for Engineering Education and Research. Innovations 2008: World Innovations in Engineering Education and Research, iNEER, Arlington, VA, U.S.A., (13p.)
12. Ieta, A., Doyle, T. E., Kucerovsky, Z., and Greason, W. D. “Enhanced Student Evaluation Software in Engineering and Science Courses", Frontiers in Education Conference 2007, Milwaukee, Wisconsin, USA, October 2007.
13. Ieta, A., Ieta, R., and Doyle, T. E. “Aggregation of Grades and Effective Grading", International Network for Engineering Education and Research (iNEER) 2007 Special Volume: “INNOVATIONS 2007: World Innovations in Engineering Education and Research