Fast and Reliable Plagiarism Detection System

Maxim Mozgovoy[1], Sergey Karakovskiy[2], and Vitaly Klyuev[3]

Abstract – **Plagiarism and similarity detection software is well-known in universities for years. Despite the variety of methods and approaches used in plagiarism detection, the typical trade-off between the speed and the reliability of the algorithm still remains. We introduce a new two-step approach to plagiarism detection that combines high algorithmic performance and the quality of pairwise file comparison. Our system uses fast detection method to select suspicious files only, and then invokes precise (and slower) algorithms to get reliable results. We show that the proposed method does not noticeably reduce the quality of the pairwise comparison mechanism while providing better speed characteristics.**

Index Terms – Plagiarism detection, similarity detection, string matching.

INTRODUCTION

A wide range of plagiarism and similarity detection systems was developed in recent years. Basically their task is to find similarities in files, which can indicate either plagiarism (in case of students' works) or code duplication (in case of a software project analysis). The quality of the system is primarily determined by the method of similarity calculation. The same method usually has a high influence on the speed of the detection. Typically, the faster is detection routine, the less precise results it provides.

In our work we study the possibility of using the combined approach: the faster (and less precise) algorithm performs the initial selection of suspicious files, and then the more reliable (and slower) routine calculates similarity ratios for file pairs. It should be noted that we concentrate on a so-called "offline" (or "hermetic") plagiarism detection, which deals with analysis of local file collections. "Online" plagiarism detection (search for similar documents in the Internet) is related more to information retrieval, so coverage can turn out to be much more important than precision.

SIMILARITY DETECTION TECHNIQUES

A good example of speed-reliability trade-off is found in the formerly popular attribute counting approach. Attribute counting systems (such as [1] and [2]) create special "fingerprints" for collection files, including metrics, such as average line length, file size, average number of commas per line, etc. The files with close fingerprints are treated as similar. Clearly, small fingerprint records can be compared rapidly, but this approach is now considered unreliable, and rarely used nowadays [3].

Modern plagiarism detection systems usually implement certain content-comparison techniques. The most popular approaches include heuristic string tiling (finding the joint coverage for a pair of files) [4, 5], and parse trees comparison [6, 7]. Usually these algorithms work for file pairs, so the comparison routine should be called for each possible file pair found in the input collection. It means that $O(f(n)N^2)$ time is required to perform the detection. Here N is the number of files in the collection, and $f(n)$ is the time needed to compare two files of length n.

Fast Plagiarism Detection System (FPDS) [8] tries to improve the algorithmic performance of plagiarism detection by utilizing a special indexed data structure (suffix array) to store input collection files. A special heuristic search routine is used to compare any given file against the whole collection at once. The complexity of the complete detection procedure is $O(nN\gamma + N^2)$, where N is the total number of files, n is the average file length, and γ is a special finely-tunable constant ($\gamma = \Omega(\log nN)$ should be used for the best performance). In terms of quality, FPDS shows the results, which are close to the ones, provided by other content-comparison systems.

TOKENIZATION

Tokenization [9] is a commonly-used technique that fights against renaming variables and changing loop types in computer programs. Simple tokenization algorithms substitute the elements of program code with single tokens. For example, all identifiers can be substituted with <IDT>, and all values with <VALUE> tokens. So, a line **a = b + 45;** will be replaced by **<IDT>=<IDT>+<VALUE>;**. Therefore, renaming variables will not help the plagiarizer.

A more advanced example of tokenization is p-match algorithm [10] that keeps track of usage of the variable names. If another file has a variable appearing in the same context, it is treated as identical.

All modern plagiarism detection systems, aimed at program code analysis, implement some tokenization-like procedure.

PLAGGIE AND FPDS

Plaggie is a recent open source content comparison-based plagiarism detection system [11]. It is based on a simplified algorithm, implemented in better-known JPlag project [5]. In its turn, JPlag develops further the idea of the Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) algorithm, used in

[1] Maxim Mozgovoy, University of Joensuu, mmozgo@cs.joensuu.fi
[2] Sergey Karakovskiy, St. Petersburg State University, sergey.karakovskiy@gmail.com
[3] Vitaly Klyuev, University of Aizu, vkluev@u-aizu.ac.jp

1-4244-1084-3/07/$25.00 ©2007 IEEE

YAP3 tool [4]. Currently it is one of the most advanced and reliable content comparison methods [12, 13].

Though the speed of detection may vary due to project-specific heuristics, the (empirically obtained) expected running time for the pure RKR-GST routine is $O(N^2 n^{1.12})$, where N is the number of files in the input collection, and n is the average file size [14].

FPDS is a rapid content comparison-based plagiarism detector, aimed at better performance at the cost of a slight quality loss [8]. Our study showed that in most cases the results, provided by FPDS is reliable enough to be used in practice. On the other hand, the underlying algorithm itself (heuristic search in the indexed data structure) brings some features that are not always acceptable. Firstly, the search routine can skip certain matches, easily detectible through RKR-GST. Secondly, the algorithm is not optimized for finding continuous matches, so the similar chunks can be uniformly spread inside the files being analyzed. This makes harder to examine the similarities visually. In contrast, Plaggie (as well as JPlag) provide detailed HTML report of the results, including similarity ratios and schemes of detected overlappings for each suspicious file pair.

The algorithm used in FPDS has one special input constant γ, standing for "typical length of a match in tokens". Large γ values lead to more mismatches, but make algorithm work faster. Smaller γ provide more precise results, but if γ becomes too small, the system will report a lot of false matches (normally we assume that one or two-token match should not be treated as plagiarism). For similarity detection in computer programs, we consider $\gamma \in [10, 30]$ to be appropriate.

The relative speed of FPDS and Plaggie can be understood from the fig. 1, demonstrating the time needed to process a sample collection of 100, 200, 300, 400, and 500 files. We have used 500 distinct files taken from Apache Tomcat project source code as our collection, then selected random smaller subsets. The complete collection size is 4.58 MB; the median file size is 6.15 KB.

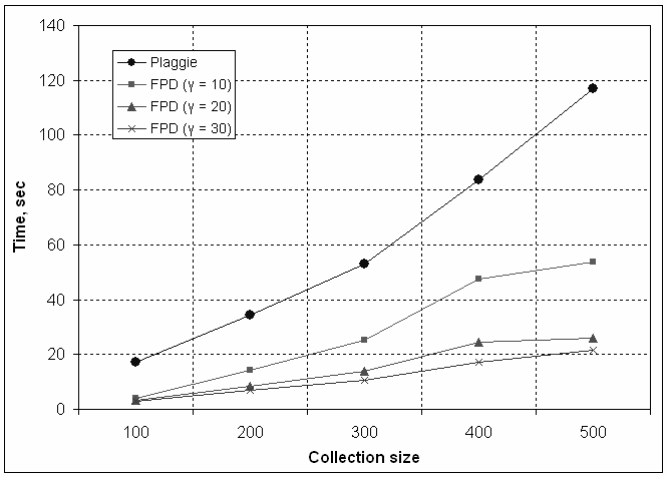

FIGURE 1
TIME REQUIRED FOR DETECTION: PLAGGIE VS. FPDS.

COMBINED APPROACH

High detection speed of FPDS with better reliability and result reporting capabilities of Plaggie suggest the idea of a combination of these two systems. Firstly, FPDS analyzes the input collection and outputs a set of suspicious files using a given similarity threshold. Then Plaggie is invoked for this set to produce a detailed and reliable report.

Technically, FPDS outputs a list of file-file similarities for every possible file pair:

```
file_i file_j sim(file_i, file_j)
```

The subset of suspicious documents we generate for further analysis by Plaggie includes every file `F`, for which

```
max(sim(F, G)) >= threshold
 G
```

It should be noted that `sim()` function is not symmetrical. If, for example, file `F` is small and file `G` is large, it can happen that `sim(F, G)` = 100% (if `G` contains `F`), while `sim(G, F)` is always less than 100%.

A combination of plagiarism detection systems we propose has to satisfy the following assumptions:

- The combined system should be noticeably faster than Plaggie.
- FPDS and Plaggie should "agree" in most cases; in other words, FPDS generally should not exclude files, treated as plagiarized by Plaggie, from the input set.

The intuitive reason for the second assumption is a close relationship between the algorithms used in both systems. Our experiments show that both these assumptions are correct. Let us consider the same five collections (100...500 files), consisting of Java source files. Table I demonstrates the agreement between Plaggie and FPDS (with γ = 20, γ = 30).

The systems mark files according to similarity thresholds ST1 and ST2. For Table I, ST1 = (0.4 for FPDS, 0.6 for Plaggie); ST2 = (0.6 for FPDS, 0.8 for Plaggie). It should be noted that since FPDS can skip matches, generally it provides lower similarity ratios that Plaggie. Therefore, we have to set higher similarity threshold for Plaggie in order to get closer reports. The percentage in parentheses shows the fraction of Plaggie-marked files, marked also by FPDS. Since these figures are enough high (81.9% on average), we can make a conclusion about reasonable agreement between the two systems. Therefore, the use of FPDS as a filter does not noticeably reduce the quality of plagiarism detection.

The speed of combined system is considerably higher than Plaggie's. The comparison of detection time graphs is shown on the fig. 2.

There are many studies investigating the nature of plagiarism but we can say nothing about the probability distribution of suspicious files. Study [16] reported that 38% of 53 students in one class plagiarized another student's assignment or allowed other students to plagiarize their assignment. After including the teacher's policy concerning academic dishonesty into the course syllabus, the number of students involved in plagiarism decreased dramatically: Only

9% of 87 students were incriminated in such incidences. No software was used to discover the aforementioned cases. Another study [17] reported the results of a survey conducted among the teachers of computing schools in Great Britain. Approximately 50% of the aforementioned schools provided the data. Responses to the question about the number of students in the school caught plagiarizing in the last academic year were ranging from a very small fraction (0.0014%) to approximately 13%.

TABLE I
AGREEMENT IN PLAGGIE AND FPDS RESULTS

Collection 1 (100 files)					
	Files marked				
Sim	Plaggie	FPDS20	FPDS30	Plaggie ∩ FPDS20	Plaggie ∩ FPDS30
ST1	23	32	19	18 (78.3%)	16 (69.6%)
ST2	14	21	11	13 (92.9%)	10 (71.4%)
Collection 2 (200 files)					
	Files marked				
Sim	Plaggie	FPDS20	FPDS30	Plaggie ∩ FPDS20	Plaggie ∩ FPDS30
ST1	72	87	66	62 (86.1%)	56 (77.8%)
ST2	51	62	46	46 (90.2%)	41 (80.4%)
Collection 3 (300 files)					
	Files marked				
Sim	Plaggie	FPDS20	FPDS30	Plaggie ∩ FPDS20	Plaggie ∩ FPDS30
ST1	92	119	89	78 (84.8%)	69 (75.0%)
ST2	63	82	60	56 (88.9%)	50 (79.4%)
Collection 4 (400 files)					
	Files marked				
Sim	Plaggie	FPDS20	FPDS30	Plaggie ∩ FPDS20	Plaggie ∩ FPDS30
ST1	118	149	115	104 (88.1%)	93 (78.8%)
ST2	80	104	78	71 (88.8%)	64 (80.0%)
Collection 5 (500 files)					
	Files marked				
Sim	Plaggie	FPDS20	FPDS30	Plaggie ∩ FPDS20	Plaggie ∩ FPDS30
ST1	138	184	141	118 (85.5%)	108 (78.3%)
ST2	92	124	91	80 (87.0%)	71 (77.2%)

We have to note, finding suspicious files using any system is not a proof of plagiarism. They have to be investigated carefully by human inspection. A technique of this inspection is not trivial. For example, on large scale tests of Plaggie for the introductory course on programming, the number of suspicious files was large and teachers had to track the students exercise by exercise using the following criteria: Similarity reported by the system was 100%, the same student pairs were under suspect, and suspicious-looking programs consisted of more than 100 lines [11]. After this complicated filtering, a number of students were asked for explanation. 80% of detected cases were confirmed.

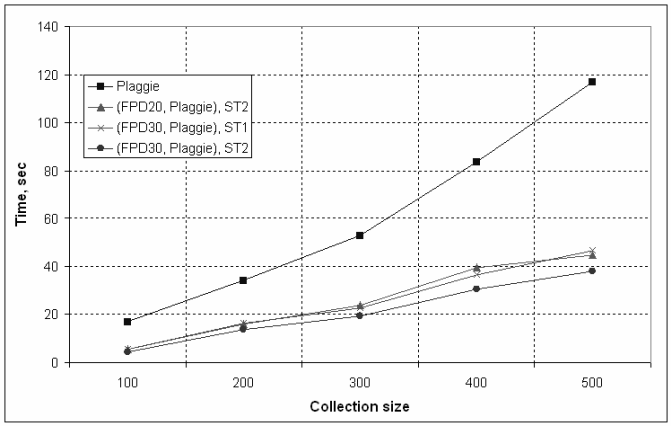

FIGURE 2
TIME REQUIRED FOR DETECTION: PLAGGIE VS. COMBINED SYSTEM.

EVALUATION

Evaluation of the systems is not a easy task. Authors and users published empirical evaluations [5, 8, 11]. Study [11] reported that Plaggie and JPlag produced practically the same results on a pool consisting of 65 student Java programs. Authors of FPDS [8] compared their system with JPlag, Sherlock and MOSS. These systems were acting as the "jury". They found: FPDS recognized 74% of suspicious files detected by the "jury". The test set consisted of 220 student Java programs. As it was noticed earlier, this result is very close to the comparison between FPDS and Plaggie: On average 81.9% of files detected by Plaggie as suspicious were detected by FPDS as well. We applied the nonparametric test of Spearman's rank correlation [15] to the data presented in Table 1. It is used to test for an association between two variables. According to the procedure, the null hypothesis is: There is no correlation between two variables; the rank correlation coefficient for the entire population is equal to zero. The alternative hypothesis claims that there is a correlation between two variables and the aforementioned rank correlation coefficient is not equal to zero. According to the results obtained, we can conclude that there is significant correlation between the data we obtained using Plaggie and FPDS ($n = 10$, $\alpha = 0.01$, $rs = 0.997$, critical value $z = 0.794$). Here rs is the sample statistic; n is the number of pairs of sample data (see columns 2 and 3, Table 1); α is the significance level, and z is a critical interval. Because the sample statistic exceeds the critical value, we made a conclusion about significant correlation. The same strong correlation is found between the data generated by Plaggie and common fractions of files marked by both systems (columns 2 and 5, Table 1). From this outcome, we can expect the same behavior of two systems when they analyze Java source code, and FPDS is a quite accurate filter for Plaggie.

CONCLUSIONS

We have developed a new fast and reliable plagiarism detection system by combining older Plaggie and FPDS projects. We have showed that the use of FPDS as a filter does

not noticeably reduce the reliability of Plaggie, but provides much better algorithmic performance than naïve file-file comparison techniques. The results were tested on a large enough set of Java source files.

As a main result, we can expect that the proposed system will filer files very fast and quite accurate. It will discard significantly the files that cannot mach criteria of plagiarism. The FPDS part is responsible for this operation. The number of files sent for a detailed investigation to the Plaggie part is small enough.

ACKNOWLEDGMENT

The authors wish to thank Kimmo Fredriksson (University of Joensuu) for reviewing the paper and providing valuable comments.

REFERENCES

[1] Grier, S., "A tool that detects plagiarism in Pascal programs", *ACM SIGCSE Bulletin*, vol. 13(1), 1981, pp. 15-20.

[2] Faidhi, J.A.W., Robinson, S.K., "An empirical approach for detecting program similarity within a university programming environment", *Computers & Education*, vol. 11(1), pp. 11-19.

[3] Verco, K.L., Wise, M.J., "Plagiarism à la mode: a comparison of automated systems for detecting suspected plagiarism", *The Computer Journal*, vol. 39(9), pp. 741-750.

[4] Wise, M.J., "YAP3: improved detection of similarities in computer program and other texts", *Proc. of SIGCSE'96 Technical Symposium*, 1996, pp. 130-134.

[5] Prechelt, L., Malpohl G., Philippsen, M., "Finding plagiarisms among a set of programs with JPlag", *Journal of Universal Computer Science*, vol. 8(11), pp. 1016-1038.

[6] Gitchell, D., Tran, N., "Sim: a utility for detecting similarity in computer programs", *Proc. of the 30th SIGCSE Technical Symposium on Computer Science Education*, 1999, pp. 266-270.

[7] Belkhouche, B., Nix, A., Hassell, J., "Plagiarism detection in software designs", *Proc. of the 42nd Annual Southeast Regional Conference*, 2004, pp. 207-211.

[8] Mozgovoy, M., Fredriksson, K., White, D., Joy, M., Sutinen, E., "Fast plagiarism detection system", *Lecture Notes in Computer Science*, vol. 3772, 2005, pp. 267-270.

[9] Joy, M., Luck, M., "Plagiarism in programming assignments", *IEEE Transactions on Education*, vol. 42(2), 1999, pp. 129-133.

[10] Baker, B.S., "On finding duplication and near-duplication in large software systems", *Proc. of the 2nd IEEE Working Conference on Reverse Engineering*, 1995, pp. 86-95.

[11] Ahtiainen, A., Surakka, S., Rahikainen, M., "Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises", *Proc. of the 6th Baltic Sea Conference on Computing Education Research*, Uppsala, Sweden.

[12] Lancaster, T., Culwin, F., "Using freely available tools to produce a partially automated plagiarism detection process", *Proc. of the 21st ASCILITE Conference*, 2004, pp. 520-529.

[13] Mozgovoy, M., "Desktop tools for offline plagiarism detection in computer programs", *Informatics in Education*, vol. 5(1), 2006, pp. 97-112.

[14] Wise, M.J., "Running Rabin-Karp matching and greedy string tiling", *Basser Department of Computer Science Technical Report*, 1994.

[15] Wessa, P., "Free statistics software", *Office for Research Development and Education*, v. 1.1.17, URL http://www.wessa.net

[16] Wiedermeier, P.D., "Preventing plagiarism in computer literacy courses", *Journal of Computing Sciences in Colleges*, vol. 17(4), 2002, pp. 154-163.

[17] Culwin, F., MacLeod, A., Lancaster, T., "Source code plagiarism in UK HE computing schools, issues, attitudes and tools", *South Bank University Technical Report*, 2001.