Towards the Detection of Cross-Language Source Code Reuse

Enrique Flores, Alberto Barrón-Cedeño, Paolo Rosso, and Lidia Moreno

Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia
{eflores,lbarron,prosso,lmoreno}@dsic.upv.es

Abstract. Internet has made available huge amounts of information, also source code. Source code repositories and, in general, programming related websites, facilitate its reuse. In this work, we propose a simple approach to the detection of cross-language source code reuse, a nearly investigated problem. Our preliminary experiments, based on character n-grams comparison, show that considering different sections of the code (i.e., comments, code, reserved words, etc.), leads to different results. When considering three programming languages: C++, Java, and Python, the best result is obtained when comments are discarded and the entire source code is considered.

Keywords: Source code reuse, cross-language source code reuse analysis, plagiarism detection.

1 Introduction

In the digital era, massive amounts of information are available causing the material from other people to be exposed to reuse. Therefore, there is high interest in identifying whether a work has been reused.

As for documents in natural language, the amount of source code in Internet is huge, facilitating the reuse of all or part of previously implemented programs. People facing similar problems are frequently tempted to source code reuse and, if no reference to the original work is included, plagiarism.[1] As a counter measure different models for the automatic detection of source code reuse (in particular plagiarism) have been developed [2, 3, 6]. The challenge of cross-language reuse detection has been approached just recently [1].

Let L_1 and L_2 be two programming languages ($L_1 \neq L_2$), we define cross-language source code reuse as the translation of (part of) a source code $a \in L_1$ into $a' \in L_2$. As for texts written in natural language [5], detecting code reuse when a translation process occurred, is even more challenging; it is very likely that a' does not represent an exact translation of a because of implementation issues.

[1] Source code reuse is often allowed, thanks to licences as those of Creative Commons (http://creativecommons.org/), but if the information related to the source is not included, plagiarism is being committed.

As far as we know the only approach that aims to detect cross-language source code reuse is that of [1]. Instead of processing source code, this approach compares intermidiate language (RTL) produced by a compiler. The comparison is in fact monolingual and compiler dependent. Unfortunately, the corpus used is not available, making the direct comparison to this approach unfeasible.

This contribution represents a preliminary work attempting to detect source code reuse among three programming languages: C++, Java and Python on the basis of Natural Language Processing techniques.

2 Model

The following levels of edition in monolingual reuse are proposed by [2]:

0. No changes in source code.
1. Represents the changes in comments and identation.
2. Includes level 1 plus changes in identifiers.
3. Groups level 2 and changes in declarations (i.e. declaring extra constants, changing the positions of declared variables, etc.).
4. Represents level 3 plus changes in program modules (i.e. merging procedures).
5. Comprises level 4 and changes in program statements (i.e. using *for* instead of *while*).
6. Represents level 5 and changes in control logic.

Our proposal aims to treat some of these levels, considering: (*i*) full code, i.e., source code and comments, for level 0; (*ii*) full code without comments (fc-without comments) for level 1; (*iii*) programming language reserved words only (fc-reserved words only) for levels 2 and 3. Additionally, three more exploratory experiments have been carried out: (*iv*) commments only (*v*) full code without reserved words (fc-without rw) and (*vi*) full code without comments and without reserved words (fc-wc-wrw).

The proposed model is divided in three steps: (*a*) Pre-processing: linebreaks, tabs and spaces removal as well as case folding; (*b*) Features extraction: character n-grams extraction, weighting based on normalised tf; and (*c*) Comparison: cosine similarity estimation.

Once a' is compared to $a \in A$, a sorted list is generated that ranks the potential sources for the suspicious program a'. The top k pairs (a', a) in the ranked list are the most similar and, therefore, more likely to be reused.

3 Experiments

Our aim is to evaluate the proposed model to detect cross-language reuse between source codes. Our toy corpus is composed of a collection of programs including source code in C++, Java and Python (the programs are formerly part of a multi-agents system). For each language a collection of programs exist that maintains a

correspondence to the programs in the other languages. The collections in C++ and Java have been partially reused. The cases Python→C++ represent real examples of cross-language reuse. The cases Python→Java represent simulated cases. Moreover, the cases C++−Java represent triangular reuse (having Python as pivot). Table 1 shows some statistics of the corpus.

Table 1. Statistics of the corpus used for the experiments.

Language	Tokens	Avg. length of tokens	Types	Types per program	Programs
C++	1,318	3.46	144	28.8	5
Java	1,100	4.52	190	47.5	4
Python	10,503	3.24	671	167.75	4

We have tested our character n-grams model considering n=$\{1, \ldots, 5\}$. Table 2 shows the average and standard deviation of the positions of that document a that is the source of a'. In the most of the experiments the best result is obtained when considering *full code* as well as *full code without comments* with the same values in both cases. The best results are obtained with $n = 3$. This is in fact the same cases as for text reuse detection [5].

Table 2. Results obtained with character 3-grams. The value represents the mean and standard deviation of the position of the source program in the ranked list.

Features	Java − C++	Python → C++	Python → Java
full code	1.00 ± 0.00	1.44 ± 0.83	1.62 ± 1.10
fc-without comments	1.00 ± 0.00	1.44 ± 0.83	1.62 ± 1.10
fc-reserved words only	1.56 ± 0.83	1.78 ± 1.02	1.75 ± 0.83
comments only	2.29 ± 1.57	2.83 ± 1.34	3.00 ± 0.67
fc-without rw	1.44 ± 0.83	1.78 ± 1.13	2.00 ± 1.32
fc-wc-wrw	1.44 ± 0.83	1.67 ± 0.94	1.44 ± 0.69

The comments in the source code has not had much impact, partly because the programmer has decided to rewrite the comment, write their own comments, or because they have not taken into account the comments when reusing the code. As a malicious programmer can modify the comments to introduce noise in the detection, it is better to ignore these sections of the program. The best results were obtained between the C++ and Java codes because they include common reused fragments from the Python implementations. Evidently, the syntax and vocabulary of C++ and Java is highly similar.

4 Conclusions and Future Work

This work is a preliminary attempt to detect cross-language source code reuse. The proposed approach is based on similarity computations at character n-grams level. The impact of comments, variable names, and reserved words of the different programming languages has been investigated. The best results are obtained when comments are ignored. This suggests that the comments can be safely discarded when aiming to determine the cross-language similarity between two programs. Presumably, the character 3-grams are able to represent programming style as in the case of documents written in natural language. No improvement was observed when weighting with tf-idf, but this could be due to the small corpus. Further experiments have to be carried out out on a larger corpus.

As future work, we identify the following avenues: (i) employing sliding windows [7] in order to compare blocks of codes, letting the location of similar fragments (for instance, a function at the beginning of a program could have been plagiarised and located at the end of another one); and (ii) applying cross-language alignment-based similarity analysis [4] that recently have given good results for texts (the necessary dictionary could be composed of reserved words).

Acknowledgments This work has been developed with the support of the project TEXT-ENTERPRISE 2.0: Text comprehension techniques applied to the needs of the Enterprise 2.0 (MICINN, Spain TIN2009-13391-C04-03 (Plan I+D+i))

References

1. Arwin, C. and Tahaghoghi, S.M.M.: Plagiarism Detection across Programming Languages. Proceedings of the 29th Australasian Computer Science Conference vol. 48, pp. 277-286. (2006)
2. Faidhi, J. and Robinson, S.: An empirical approach for detecting program similarity and plagiarism within a university programming environment. Comput. Educ., vol. 11, pp. 11-19. (1987)
3. Jankowitz, H. T.: Detecting plagiarism in student pascal programs. The computer journal, vol. 31(1). (1988)
4. Pinto, D., Civera, J., Barrón-Cedeño, A., Juan, A., and Rosso, P. A statistical approach to crosslingual natural language tasks. Journal of Algorithms, vol. 64(1), pp. 51-60. (2009)
5. Potthast M., Barrón-Cedeño A., Stein B., Rosso P.: Cross-Language Plagiarism Detection. In: Languages Resources and Evaluation. Special Issue on Plagiarism and Authorship Analysis, vol. 45(1). (2011)
6. Rosales, F., García. A., Rodríguez, S., Pedraza, J. L., Méndez, R., and Nieto, M. M.: Detection of plagiarism in programming assignments. IEEE Transactions on Education, vol. 51(2), pp. 174-183. (2008)
7. Stamatatos, E.: Intrinsic Plagiarism Detection Using Character n-gram Profiles. In Proc. SEPLN'09, Donostia, Spain, pp. 38-46. (2009)