# Cross-Language Identification of Similar Source Codes based on Longest Common Substrings

René Arnulfo García-Hernández
Autonomous University of the State of Mexico
Santiago Tianguistenco, San Pedro Tlaltizapan
State of Mexico, Mexico
renearnulfo@hotmail.com

Yulia Ledeneva
Autonomous University of the State of Mexico
Santiago Tianguistenco, San Pedro Tlaltizapan
State of Mexico, Mexico
yledeneva@yahoo.com

## ABSTRACT
In this paper, we describe the system developed by Autonomous University of the State of Mexico (in Spanish, UAEM) for the cross-language detection of source code re-use (CL-SOCO) task of FIRE-2015. The aim of the CL-SOCO task is to detect the most similar code pairs from C to Java languages. Since Java and C share most of the lexical and syntactical information, we preprocess few of the most frequent Java and C instructions to try to unify both languages. Then, the CL-SOCO task can be seen as monolingual detection, as the previous task of SOCO. Since our approach was ranked well in the previous participation in SOCO, we decide just preprocess both source codes without re-training. According to the test evaluation our approach was ranked in the fourth position close to the third ranking.

## Categories and Subject Descriptors
H.3 [**Information Storage and Retrieval**]: H.3.1 Content Analysis and Indexing; H.3.3 Information Search and Retrieval; H.3.4 Systems and Software

## General Terms
Algorithms, Measurement, Performance, Experimentation, Languages

## Keywords
Source Code Re-use, Longest Common Substrings, Similar Codes, Java Code Re-use, C Code Re-use, Cross-Language Source Code Detection.

## 1. INTRODUCTION
Even though is common to find a lot of web pages showing source codes in different languages, the source code is the result of an intellectual effort, for such reason, it is protected by copyright laws. Normally, the source code in the Web is presented in short fragments with tutorial proposes. However, re-using source code of works brings economic problems for the author and legal problems for whom make the act. Some automatic tools [5][6] have been developed to assist with the problematic of the re-use monolingual detection of source code. Nevertheless, an expert programmer can easily translate a C program to Java language making invisible for monolingual source code re-use detection tools. In the CL-SOCO task [1], the aim is to propose approaches to deal with the cross-language re-use from C to Java. In this paper, our participation UAEM (Autonomous University of the State of Mexico) in the CL-SOCO task is described.

## 2. Proposed Approach
Since Java and C languages have similar grammar, we propose to preprocess both languages in order to unify them. In this manner, the cross-language detection problem can be seen as mono-language problem. In the previous participation in SOCO task [2], our approach was ranked well; therefore, we use the same system. In preliminarily experiments with the first-provided training collection, our system performs good results but some of the false-positives results in our opinion were true-positives. In this case, we decide do not re-training our system [3] adapting only the preprocessing phase.

Our system (UAEM) used for the detection of source code re-use is divided into four phases.

### 2.1 Preprocessing phase
In the first phase, only the lexical items (like {,},(,),+,*,;.etc.) of each source code are separated with a whitespace and more than one whitespace is removed. The next Java instructions are translated to C:

```
String.charAt(Number) → String [Number].
System.exit → exit
System.out.print(f|l)→ printf.
```

In the case of C instructions:

```
strcmp(Id1,Id2) → Id1 == Id2
strcpy(Id1,Id2) → Id1 = Id2
```

The result of this phase is a string of tokens of the source code.

### 2.2 Similarity measure phase
In the second phase, for each source code given as a string, the similarity measure with respect to the other source codes is obtained. The sum of the different lengths of the longest common substrings between the two source codes (normalized to the length of the longest code) is used as the similarity measure. For this phase, we used the algorithm described in [4]. The similarity between two codes with the same language is defined as zero since for this task it is not interesting.

### 2.3 Ranking phase
In the third phase, a set of parameters that allow later the identification of cases of re-use is obtained using comparisons done in the previous phase. The parameters obtained are: the value of the DISTANCE (1 - similarity), the RANKING of the distance (rank order of the most similar), the GAP that exists with the next closest code (it is only calculated for the first 10

closest codes) and, using the maximum gap between the 10 most closer codes, the codes that are (B)efore or (A)fter the maximum gap (RELATIVE DIFFERENCE) are labeled. The result of the third phase is a matrix where each row represents a comparison of a source code with other codes (columns) and each cell represents a pair of source codes in both directions.

## 2.4 Re-use decision phase

Even though in the CL-SOCO task the re-use was committed only in the direction from C to Java, we believe that for taking the decision there must be evidence in both directions. For taking the decision, a source code pair X↔Y will be a re-use case, if there is evidence of re-use in both directions, it means, X→Y and Y→X. A re-use case exists when the DISTANCE is less than 0.45 or the GAP is greater than 0.14, but also it is important that one of the additional conditions is achieved. The first condition is that the RANKING must be, at least, in the second position and, the second condition, that the label of the RELATIVE DIFFERENCE must be B. The first run was processed with above conditions. However, in some cases the evidence in one direction was very high and in the other direction was almost reliable. In the second run, if there were not high evidence of re-use in one direction, then the pair can be considered as re-use case whether at least one of the both codes has the RANKING of 1, the RELATIVE DIFFERENCE of B, and the GAP greater than 0.1.

# 3. Training corpus

The first training corpus consists of 599 source codes in Java and 599 sources codes in C, where the re-use where committed from the file pair *id.c* to *id.java*. In our preliminarily evaluations, the approach presents good results but some false-positives results were confusing since according to our judgment were actually true-positives. In this moment, we decide to use the values presented above that were obtained with the training of the SOCO task. However, it is worth to say that the second version of the training corpus was provided without these mistakes, but because for time reasons we do not tune our system.

# 4. Testing experiments

In contrast, with the previous SOCO task, the test corpus of CL-SOCO was smaller with 79 programs in java and 79 programs in C. Table 1 shows the evaluation of the five systems proposed for the CL-SOCO task. Our first run of the system is ranked in the fourth place according to F1 measure, very close to third place with a difference of 0.001. The difference with the best system is of .033.

**Table 1. Test evaluation with the five participants.**

| Rank | Team-Run | F1 | Precision | Recall |
|---|---|---|---|---|
| 1 | UAM-C_run1 | 0.772 | 0.988 | 0.634 |
| 2 | Palkovskii_run1 | 0.752 | 1.000 | 0.603 |
| 3 | PES_BSec_run2 | 0.740 | 1.000 | 0.588 |
| 4 | UAEM_run1 | 0.739 | 0.975 | 0.595 |
| 5 | Palkovskii_run2 | 0.724 | 0.962 | 0.580 |
| 6 | UAEM_run2 | 0.709 | 1.000 | 0.550 |
| 7 | UAEM_run3 | 0.703 | 1.000 | 0.542 |
| 8 | PES_BSec_run3 | 0.697 | 1.000 | 0.534 |
| 9 | UAM-C_run2 | 0.687 | 0.620 | 0.771 |
| 10 | PES_BSec_run1 | 0.683 | 1.000 | 0.519 |
| 11 | UAM-C_run3 | 0.655 | 0.496 | 0.962 |
| 12 | CLSCR_run1 | 0.611 | 0.952 | 0.450 |

# 5. Conclusions and future work

In this paper, a new system for detecting a cross–language source code re-use is described. The proposed system works in four phases. The preprocessing phase is very interesting since it does not require sophisticated processes or dictionaries, making the execution of this phase very fast. It is worth noting that all phases of our system work with words, making the process faster than when working with characters. The second phase introduces a new measure based on the different lengths of longest common substrings between the pairs of source codes which outperform LCS. The third phase considers other parameters derived from the LCSs measure. These parameters were tuning with the previous task of SOCO. In this sense, our approach is robust since get good results in both tasks.

In the future, we expect to adjust our system in the preprocessing phase and the rules of the third phase with the correct training corpus.

# 6. REFERENCES


[1] E. Flores, P. Rosso, L. Moreno, and E. Villatoro-Tello: PAN@FIRE 2015: Overview of CL-SOCO Track on the Detection of Cross-Language SOurce COde Re-use. In Proceedings of the Seventh Forum for Information Retrieval Evaluation (FIRE 2015), Gandhinagar, India, 4-6 December (2015)

[2] E. Flores, P. Rosso, L. Moreno, and E. Villatoro-Tello, 2014. PAN@FIRE 2014: Overview of SOCO Track on the Detection of SOurce COde Re-use. In *Proceedings of the Sixth Forum for Information Retrieval Evaluation* (FIRE 2014), Bangalore, India.

[3] R.A. García-Hernández, Y. Ledeneva, 2014. Identification of Similar Source Codes base on Longest Common Substrings. In *Proceedings of the Sixth Forum for Information Retrieval Evaluation* (FIRE 2014), Bangalore, India.

[4] R.A. García-Hernández, J. Martínez-Trinidad and J. Carrasco-Ochoa, 2006. A new algorithm for fast discovery of maximal sequential patterns in a document collection. Computational Linguistics and Intelligent Text Processing, LNCS 3878, Springer, 514-523, Mexico.

[5] L. Prechelt, G. Malpohl and M. Phlippsen, 2000. JPlag: Finding plagiarism among a set of programs. Technical Report, Universität Karlsruhe, Germany.

[6] A. Aiken. 1998. MOSS (Measure Of Software Similarity) plagiarism detection system., University of Berkeley, CA.