Cross-Language Source Code Re-Use Detection

Enrique Flores[1], Alberto Barrón-Cedeño[2], Lidia Moreno[1], and Paolo Rosso[1]

[1] Universitat Politècnica de València, Spain
[2] Talp Research Center, Universitat Politècnica de Catalunya, Spain
{eflores,lmoreno,prosso}@dsic.upv.es, albarron@lsi.upc.edu

Abstract. Repositories, forums, or websites like Rosettacode.org make a vast amount of source codes available. With the growth of the Web, contents' re-use has increased. Source code re-use detection allows to spot potential instances of re-use. In the recent years, source code re-use detection has been tackled mainly using compilers. When we deal with a cross-language source code re-use scenario, the detection is restricted to the languages supported by the compiler. Assuming a source code as a piece of text with its syntax and formal structure, we aim at applying models for text re-use detection to source code. In this paper we compare models which do not rely on external resources for measuring cross-language similarity —cross-language character n-grams, pseudo-cognateness, word count ratio—, against corpora-dependent models —cross-language explicit semantic analysis and cross-language alignment-based similarity analysis. In our experiments, a combination of cross-language character 3-grams and pseudo-cognateness performed better than the corpora-dependent models. The latter models improved their performance when exploiting larger corpora. All in all, the applied models showed to be able to retrieve both re-used and co-derived source codes.

Keywords: Cross-Language Re-Use Detection, Source Code, Plagiarism

1 Introduction

Manual retrieval of re-used documents from a large collection is a difficult task; manual retrieval from the Web is unfeasible. Automatic re-use detection systems can easily retrieve potential re-used candidates on the fly. Over the last few years, re-use detection has caused a great interest promoting international tracks [2] for text re-use and for source code re-use.[3]

Detection of re-use has been tackled as an information retrieval (IR) problem. IR looks for related documents and re-use detection focuses on those documents that have been re-used (partially or completely). As IR systems, re-use detection systems can be divided in three phases: *(i)* query processing, the query is pre-processed in the same way as the reference collection; *(ii)* comparison against the collection, the similarity between the query and the documents in the collection

[3] http://pan.webis.de/ and http://www.dsic.upv.es/grupos/nle/soco/

is calculated; and *(iii)* ranking of results, where the collection is sorted according to the documents' similarity to the query, composing a relevance ranking.

In this paper we investigate models for cross-language source code re-use detection. Considering source code documents as simple text containers allows us for borrowing techniques for cross-language text re-use detection [3]. As far as we know, few research work exists that face up cross-language source code re-use detection [4,5]. We looked for instances of cross-language source code re-use on the most used programming languages nowadays: C, Java and Python.

The rest of the paper is organised as follows. In Section 2 we overview different approaches to detect source code re-use. In Section 3, we describe the corpora we built out of the Rosettacode website. Section 4 describes the five models we compare in this work. In Section 5, we discuss the experiments we carried out and the obtained results. Finally, in Section 6 we draw conclusions and propose future work.

2 Related Work

Text re-use detection has been widely explored [6,7,8] from different perspectives. Monolingual analysis has been tackled with a wide range of approaches from word n-grams [9,10] or fingerprinting [8] up to thesaurus-based approaches [11]. Cross-language text analysis has been mainly tackled with four different approaches: *(i)* dictionary-based [12]; *(ii)* syntax-based [13]; *(iii)* comparable corpora [14]; and *(iv)* parallel corpora [15]. An overview can be found in [6].

When speaking about natural language text, comparable documents are those written on the same topic and written in different languages [16]. Parallel documents are two documents which are a translation of each other [17]. In the framework of computer programs, we consider that two codes are comparable if they solve the same problem. As expected, comparability does not imply re-use. When a program is an (near-) translation of a code in a different language, it can be considered as parallel. By considering this parallelism between natural-language and programming text, both with a well-defined syntax and vocabulary, we can adapt the aforementioned models to be applied to source code re-use detection.

Automatic source code re-use detection has been approached from two main perspectives: attribute-counting and structure-based [3]. Attribute-counting uses characteristics as number of identifiers in a source code [18], average nesting depth [19] or times of a function is called [20]. Structure-based approaches have been more widely explored than attribute-counting because they can detect re-use even after the source code has been altered [21]. In all these approaches, the process of detection of re-use is carried out in three steps: *(i)* an initial pre-processing that removes comments, spaces, and punctuation marks; *(ii)* a representation of the source code using its structure; and *(iii)* a comparison of the representations is made to give a similarity value. In order to represent the code's structure, [21] proposed to use the bifurcation statements of the source code as comparison unit and the kind of bifurcations to determine their simi-

larity. This work inspired other approaches whose tools, such as YAP3 [22] and JPlag [23], use also efficient string matching algorithms for finding similar code fragments.

Another approach that requires less knowledge about the programming languages is based on Winnowing algorithm [24]. Initially, it selects the reserved words, delimiters, operators of the programming language and then, fragments of source codes are represented with hashes. Whereas this model is highly efficient, it is weak against editions: modifying one single character in a string dramatically alters its resulting hash value, causing potential cases of reuse to go undetected.

Other proposals use the output of a compiler to represent the structure of the source code. In [25], the authors compare the syntax tree generated during the compiler's parsing process. The syntax tree is converted into a fingerprint for locating similar fragments. Another compiler-based proposal is based on the comparison of the program's dependencies graph [26]. This approach is powerful against changes in the structure of the source code without changing its behaviour (code refactoring). It consists of using the dependency graph created by the compiler and then looking for isomorphic sub-graphs between the dependency graphs that represent the source codes. This approach is not suitable for near real-time applications because search for isomorphic sub-graphs is a costly operation (NP-hard problem). These proposals are compiler-dependent so they only can be used in the programming languages supported by the compiler.

In [4], an interesting compiler-dependent approach is proposed to cross-language source code detection: programs are compiled to generate intermediate code. As some suites exist that are able to compile code from different languages (e.g. the GNU Compiler Collection), they analyse two codes in different languages by comparing their common-language version in an intermediate language. However, this approach is not language-independent, as it needs a common compilation framework that includes the required languages.

In this work we compare different models for cross-language source code reuse detection ranging from those that require parallel and comparable corpora for training to models that do not require any corpus.

3 Corpora

Rosettacode.org is a website that presents solutions to the same task in as many different programming languages as possible. In a snapshot taken on Feb. 27 2012, there were 600 pages of solutions written in C, 598 in Python, 448 in Java, 403 in C#, and 370 in C++. We took implementations in the three most used languages: C, Java, and Python. As many solutions have more than one implementation, we consider all the possible combinations of the same solution to compose a comparable corpus[4].

[4] The C-Java, Java-Python, and C-Python partitions of the corpus can be downloaded from `http://rosettacode.org/`

Table 1. Number of comparable and parallel source code pairs per each pair of programming languages.

	C-Java	Java-Python	C-Python
Comparable	959	1588	1408
Parallel	335	335	335

In order to simulate a parallel and a comparable scenario, we used automatic source code translators to create a parallel corpus. We construct the parallel corpus as follows: all the comparable source codes written in C language were translated into Java using *C++ to Java Converter*[5]; and then the translated Java source codes were translated into Python using *java2python*[6]. After translation, comments introduced by the translators were removed. Note that *java2python* generates a syntactically (almost) equal valid Java source codes, whereas *C++ to Java Converter* is able to rephrase and refactor source code if necessary. This implies differences between translations in terms of source code length. The parallel corpus can be obtained applying the described process to the original Rosetta partition.

Table 1 shows the amount of comparable and parallel source code pairs between each pair of programming languages. We divide the documents into training and text collections: 70% is used to train the respective retrieval model and 30% for test. As aforementioned, CL-ESA requires the comparable training collection as index documents, whereas CL-ASA requires the parallel training collection to train bilingual dictionaries and length models. The rest of models do not require any training resource.

4 Models

In this paper our aim is to apply IR models that previously performed well on cross-language text re-use detection [6], [27] to cross-language source code re-use detection. We considered both models that require external resources and models that do not. Cross-Language Explicit Semantic Analysis (CL-ESA) has shown good performance in a comparable scenario and Cross-Language Alignment-based Similarity Analysis (CL-ASA) has worked accurately in a parallel scenario [6]. Cross-Language Character 3-grams, Pseudo-Cognateness and Word Count Ratio, which do not rely on any resource, have shown to be worth considering to assess similarities within comparable corpora [27]. Below, we present our adaptations to these models to the source code scenario.

4.1 Cross-Language Explicit Semantic Analysis (CL-ESA)

The model consists of calculating the similarities between two source codes comparing them against a comparable corpus to build two vectors of similarities.

[5] `http://www.tangiblesoftwaresolutions.com/Product_Details/CPlusPlus_to_Java_Converter_Details.html`

[6] `https://github.com/natural/java2python`

A source code d (d') written in the programming language L (L') is compared against a source code collection C (C') written in language L (L') to generate a vector of similarities $\overrightarrow{d}$ ($\overrightarrow{d'}$). As C and C' are a comparable corpus, the i-th element of $\overrightarrow{d}$ and $\overrightarrow{d'}$ represents their similarity to a comparable source code pair $\{c_i, c'_i\}$. Therefore, the resulting vectors can be compared to each other to indirectly estimate the similarity between d and d'. Note that the comparison against the whole collection can be based on traditional information retrieval techniques, such as an inverted index. This makes the comparisons affordable, even for large collections. At the end, the vectors of similarities are compared with a similarity function (Equation 1). Although the "semantic" component of the model may be not too obvious when CL-ESA is applied to detect cross-language source code re-use, the model allows to detect when source codes are written in semantic similar ways (e.g. *for* and *while* statements).

$$\overrightarrow{d} = \{sim(d, x) \forall x \in C / C \in L\} \qquad \varphi(d, d') = sim(\overrightarrow{d}, \overrightarrow{d'}) \tag{1}$$

4.2 Cross-Language Alignment-based Similarity Analysis (CL-ASA)

CL-ASA is based on statistical machine translation principles, where a translation model and a language model (among others) are combined to generate the most likely translation of a text [28]. In CL-ASA, the language model is substituted by a length model, which determines the likelihood a code of being a translation of one another according to its expected length. The length model, originally proposed by [29], is defined as:

$$lf(d, d') = e^{-0.5\left(\frac{len_d / len_{d'} - \mu}{\sigma}\right)} \tag{2}$$

where μ and σ are the mean and standard deviation of the lengths between translations of source codes from language L into L'. Table 2 shows the values of μ and σ for the programming language pairs we consider. This parameters have been estimated from the parallel training collection.

Table 2. Estimated length factors for each language pair measured in tokens. A value of $\mu > 1$ implies $|d| < |d'|$ for d and its translation d' in terms of characters.

Parameter	C->Java	Java->C	Java->Python	Python->Java	C->Python	Python->C
μ	0.92469	1.46322	1.00313	1.00824	0.98885	1.56418
σ	0.50461	3.08911	0.07742	0.08280	0.79928	2.59504

We use the translation model to determine if the tokens in code d are valid translations from the tokens in code d'. It requires a statistical bilingual dictionary, usually estimated from a sentence-aligned parallel corpus. For source code we consider whole programs as parallel units, whose vocabulary is limited to

reserved words, operators, and identifiers. In addition, all identifiers are substituted by a generic identifier. To estimate the translation probabilities, we use the IBM 1 alignment model [28].

In [30] it is proposed an adaptation to the translation model, originally intended to handle sentences, to estimate the similarity between entire documents. It is defined as:

$$w(d|d') = \sum_{x \in d} \sum_{y \in d'} p(x, y) \ , \tag{3}$$

where $p(x, y)$ represents the entries in the bilingual dictionary. This adaptation is not a probability, as it is not ranged in $[0, 1]$, and larger documents produce higher similarity values. Similarity between d and d' results of the combination of this adapted translation model and the length model:

$$\varphi(d, d') = lf(d, d') \cdot w(d|d') \ , \tag{4}$$

which downgrades the negative impact of a range-less translation model.

We tried four variants of CL-ASA: every potential translation for a given entry as well as only the k most likely up to filling a 20% and 40% of the probability mass; the fourth variant consists of considering the training corpus for learning the statistical dictionary. We used a conversion table[7] of the basic instructions of programming languages to learn the statistical dictionary. We considered the model that uses the 0.4% most likely of being translation from the dictionary because it achieved a slightly better performance than the other CL-ASA versions.

4.3 Cross-Language Character 3-grams (CL-C3G)

Character n-grams have shown good accuracy for monolingual and cross-language information retrieval [13]. Character n-grams also achieved noticeable results when applied to cross-language source code re-use detection [31].

The pre-processing consists of removing line feeds, tabs, and spaces; characters are also case-folded. The source code is then split into contiguous overlapping sequences of 3 characters, which are weighted by term frequency (tf). The similarity between programs is estimated using the cosine measure.

4.4 Pseudo-Cognateness (COG)

Cognates are defined as pairs of tokens of different languages which, usually share some phonological or orthographic properties. As a result, they are likely to be used as mutual translations [32]. We consider as cognate candidates to those tokens with at least one digit, and the tokens composed by letters with at least four characters. Using pseudo-cognates, we determine how two pieces

[7] The conversion table was extracted from a Wikipedia article at `http://en.wikipedia.org/wiki/Comparison_of_programming_languages_%28basic_instructions%29`

of source code are related. The pre-processing consists of extracting from the source code the cognate candidates and case folding them. Then, the similarity is calculated as:

$$\varphi(d, d') = \frac{c}{(m+n)/2} \quad , \tag{5}$$

where parameters m and n are the number of cognate candidates in each source code; then match these candidates so as to obtain the largest possible number c of pairs of cognates, without using the same token twice.

4.5 Word Count Ratio (WCR)

Word Count Ratio is the simplest of the considered models. Firstly, tokens are extracted from the source codes and delimiters (e.g. semicolon, brackets, ...) are discarded . Then, each source code is represented by the number of tokens it has. The similarity computed as the length ratio between the shorter and the longer source code in number of tokens.

$$\varphi(d, d') = \frac{min(len(d), len(d'))}{max(len(d), len(d'))} \quad , \tag{6}$$

5 Experiments

In our evaluation we compare the models in isolation as well as combinations of the three resource-free models in a ranking task. CL-ASA has not been combined because not being normalised, and CL-ESA due to its poor results. Two experiments are conducted on two test collections over all possible programming language pairs. In total, more than five million similarities are computed with each model.

Our experimental setup is as follows: d is a query source code from a test collection D, D are the documents aligned with those in D', and d' the document that is aligned with d. Given d, all documents in D' are ranked according to their cross-language similarity to d; the retrieval rank of d' is recorded. Ideally, d' should be at the top of the ranking. The experiments have been repeated using 5-fold cross-validation on both test collections, averaging the results.

In the experiments we simulated the scenario of cross-language source code re-use when a source code is given and the goal is to retrieve its aligned source code pair from a source code collection. In the parallel scenario, the translated source code pair is the one to be retrieved, whereas in the comparable scenario, is a source code solution for the same problem. The results of the experiment are shown in Figure 1 as recall-over-rank plots. We observe a difference between the parallel and the comparable scenario in terms of recall. Comparable source code pairs could be different implementations and might not share code fragments. Nevertheless, the models showed to be able to retrieve co-derived source codes in this comparable scenario, i.e., related source codes.

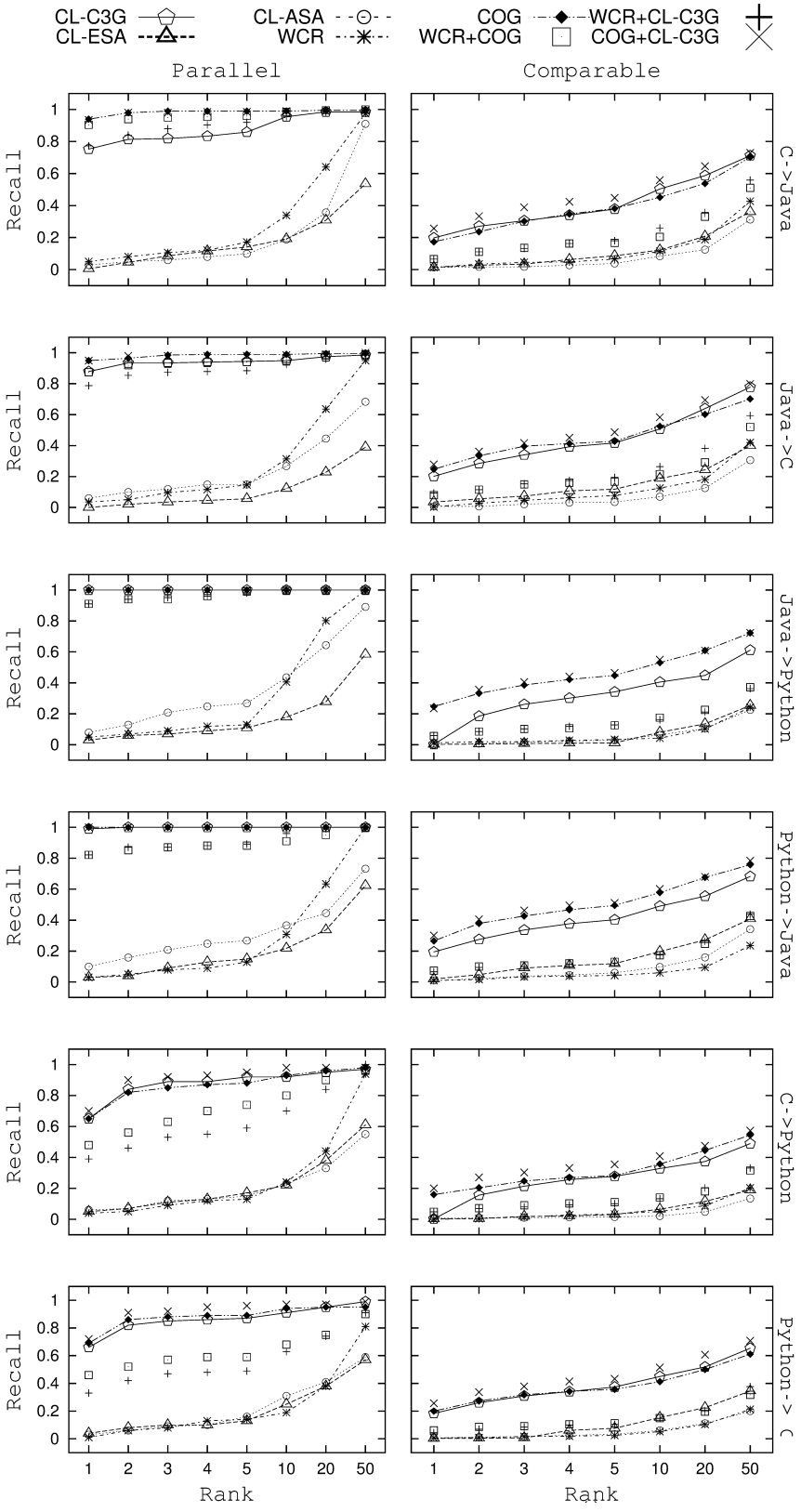

Fig. 1. Comparison of the models in a comparable and parallel scenario in terms of recall.

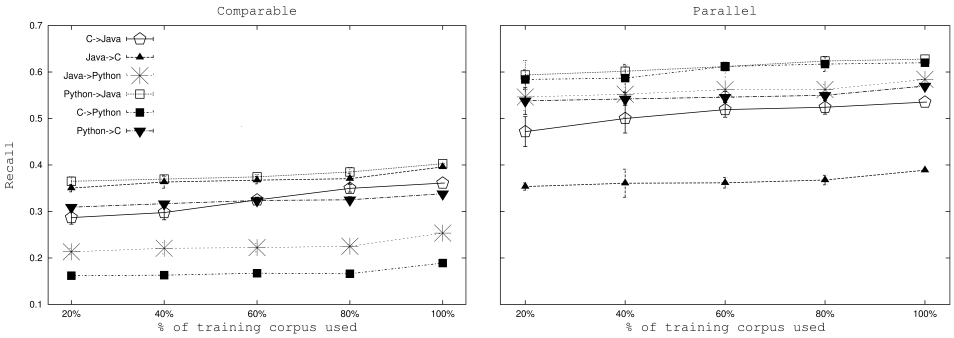

Fig. 2. Comparison of different size of training corpora in the CL-ESA model in terms of recall in the top 50 for all the programming language pairs.

The combination of COG and CL-C3G shows the best performance in most language pairs and scenarios, also outperforming their application in isolation. Combining the resourceless models with WCR the performance of these slightly decreases.

CL-C3G and COG show nearly-perfect performance in the Java–Python parallel scenario. The reason could be in the *java2python* translator, which produces almost syntactically-identical translations. These models perform well when the syntax, structure, and vocabulary are highly similar. Another evidence of this phenomenon is the length factor parameters estimated in Section 4.2, where μ is closer to 1 and the deviation is small. That is, the translations have an almost equal number of tokens than the translated source codes.

In general, the corpora-based models achieve perform worse than the resourceless models. This low performance may be caused by the small amount of programs used for training. In order to support this hypothesis, we trained CL-ESA models with increasing amounts of comparable codes. Figure 2 shows a slight trend of growing performance for all programming language pairs and scenarios as a bigger corpus is used for training. We studied whether these changes are statistically significant, by comparing the model using the 20% and the 100% of the training corpus. As Table 3 shows, significant differences exist in most comparable scenarios. These figures are in agreement with our hypothesis: the bigger the comparable corpus used by CL-ESA, the better its performance.

Table 3. Results of the significance Student's t-test in CL-ESA model between considering 20% and 100% of training corpus. A p-value < 0.05 means the difference is significant.

	C->Java	Java->C	Java->Python	Python->Java	C->Python	Python->C
Parallel	0.1141	**0.0003**	0.3863	0.2775	0.4201	0.2337
Comparable	**0.0015**	0.3581	**0.0442**	**0.0000**	**0.0000**	**0.0000**

6 Conclusions and Future Work

Source code re-use detection is important for academia and software companies. In fact, a student or programmer can retrieve a piece of source code from the Web and use it as their own without acknowledging the source. Even, they can translate into another programming language. It is also interesting for searching for similar source codes that solve the same problem in a different programming language.

In this paper we analyse the performance of different cross-language similarity models to uncover potential cases of source code re-use. The considered IR models were borrowed from natural-language text re-use detection, where they had previously shown good results. We consider models that either rely or not on external resources and apply them on two scenarios: looking for instances of translated source code and looking for source codes that solve the same problem in a different programming language.

The best performance was obtained by combining a model based on pseudo-cognates and one based on character 3-grams; none of which require any external resource. When looking for source code translations, this combination adequately retrieved the translation of a source code with nearly-perfect performance. When looking for codes approaching the same problem in different languages the models retrieved those source codes that were re-used among the different solutions. Although the models that require external resources performed worst in general, further experiments showed that a major extent of the resources would allow for significantly better results.

As future work, we plan to apply other cross-language similarity models, including latent semantic analysis, as well as exploring more complex combinations among the different models.

Acknowledgements

This work has been supported in part by Universitat Politècnica de València, WIQ-EI (IRSES grant n. 269180) and DIANA-APPLICATIONS (TIN2012-38603-C02- 01) project. The work of the fourth author is also supported by VLC/CAMPUS Microcluster on Multimodal Interaction in Intelligent Systems. The work of the second author was funded by the Spanish Ministry of Education and Science (TACARDI project, TIN2012-38523-C02-00).

References

1. Chapman, K.J., Lupton, R.: Academic dishonesty in a global educational market: a comparison of Hong Kong and American university business students. International Journal of Educational Management, 18(7), pp. 425–435 (2004)
2. Potthast, M., Barrón-Cedeño, A., Stein, B., Rosso, P.: An Evaluation Framework for Plagiarism Detection. In: Proc. of the 23rd Int. Conf. on Computational Linguistics, ACL, pp. 997–1005 (2010)
3. Clough, P.: Plagiarism in natural and programming languages: an overview of current tools and technologies. Dept. of Computer Science, University of Shefield, UK., CS-00-05, pp. 1–31 (2000)

4. Arwin, C., Tahaghoghi, S.: Plagiarism detection across programming languages. In Proc. 29th Australasian Computer Science Conference, 48, pp. 277–286 (2006)
5. Flores, E., Barrón-Cedeño, A., Rosso, P., Moreno, L.: Towards the detection of cross-language source code reuse. In Natural Language Processing and Information Systems, LNCS, vol. 6716, pp. 250–253, Springer Berlin Heidelberg (2011)
6. Potthast, M., Barrón-Cedeño, A., Stein, B., Rosso, P.: Cross-Language Plagiarism Detection. In: Language Resources and Evaluation, Special Issue on Plagiarism and Authorship Analysis, 45(1), pp. 45–62 (2011)
7. Stamatatos, E.: Plagiarism detection using stopword ngrams. Journal of the American Society for Information Science and Technology, 62(12), pp. 2512–2527 (2011)
8. Brin, S., Davis, J., García-Molina, H.: Copy detection mechanisms for digital documents. In Proc. of the Int. Conf. on Management of Data, ACM SIG-MOD, 24(2), pp. 398–409 (1995)
9. Stamatatos, E.: Intrinsic plagiarism detection using character n-gram profiles. In. Proc. of the 3rd Int. Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse, pp 38–46 (2009)
10. Barrón-Cedeño, A., Rosso, P. On Automatic Plagiarism Detection based on n-grams Comparison. In: Proc. 31st European Conf. on Information Retrieval, LNCS, vol. 5478, pp. 696–700, Springer-Verlag (2009)
11. Kang, N., Gelbukh, A., Han, S.: PPChecker: Plagiarism Pattern Checker in document copy detection. In Text, Speech and Dialogue, LNAI, vol. 4188, pp. 661–667, Springer Berlin Heidelberg (2006)
12. Gupta, P., Barrón-Cedeño, A., Rosso, P.: Cross-language high similarity search using a conceptual thesaurus. In Proc. 3rd Int. Conf. of CLEF Initiative on Information Access Evaluation meets Multilinguality, Multimodality, and Visual Analytics, LNCS, vol. 7488, pp. 67–75, Springer-Verlag (2012)
13. Mcnamee, P., Mayfield, J.: Character N-Gram Tokenization for European Language Text Retrieval. Inf. Retr., 7(1-2), pp. 73-97 (2004)
14. Potthast, M., Stein, B., Anderka M.: A Wikipedia-Based Multilingual Retrieval Model. In 30th European Conference on IR Research, LNCS, vol. 4956, pp. 522-530, Springer Berlin Heidelberg (2008)
15. Barrón-Cedeño, A., Rosso, P., Pinto, D., Juan, A.: On Cross-Lingual Plagiarism Analysis Using a Statistical Model. In ECAI 2008 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 08), pp. 9-13 (2008)
16. Potthast, M., Stein, B., Anderka, M.: A Wikipedia-based multilingual retrieval model. Advances in Information Retrieval, pp. 522–530, Springer Berlin Heidelberg (2008)
17. Steinberger, R., Pouliquen, B., Widiger, A., Ignat, C., Erjavec, T., Tufis, D., Varga, D.: The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. In: Proc. of the 5th Int. Conf. on Language Resources and Evaluation (2006)
18. Halstead, M.: Naturals laws controlling algorithm structure?. SIGPLAN Notices, 7(2) (1972)
19. McCabe, T.J.: A Complexity Measure. IEEE Trans. Software Eng. 4(2),pp. 308–320 (1976)
20. Selby, R.: Quantitative studies of software reuse. Software Reusability, 2, ACM, pp. 213–233 (1989)
21. Whale, G.: Software metrics and plagiarism detection. Journal of Systems and Software,13(2), pp. 131–138 (1990)
22. Wise, M.: Detection of similarities in student programs: Yaping may be preferable to plagueing. Proc. 23th SIGCSE Technical Symposium (1992)

23. Prechelt, L., Malpohl, G., Philippsen M.: Finding plagiarisms among a set of programs with JPlag. J. of Universal Computer Science, 8(11), pp. 1016–1038 (2002)
24. Marinescu, D., Baicoianu, A., Dimitriu, S.: A Plagiarism Detection System in Computer Source Code. Int. J. Computer Science Research and Application, 3(1), pp. 22–30 (2013)
25. Chilowicz, M., Duris, E., Roussel, G.: Syntax tree fingerprinting for source code similarity detection. 17th Int. Conf. Program Comprehension, IEEE, pp. 243–247 (2009)
26. Krinke, J.: Identifying similar code with program dependence graphs. Proc. 8th Conf. Reverse Engineering, IEEE, pp. 301-309 (2001)
27. Barrón-Cedeño, A., Lestari-Paramita, M., Clough, P., Rosso, P.: A Comparison of Approaches for Measuring Cross-Lingual Similarity of Wikipedia Articles. Proc. 36th European Conf. on Information Retrieval, Springer-Verlag (In Press), Springer-Verlag (2014)
28. Brown, P., Pietra, V., Pietra, S., Mercer, R.: The mathematics of statistical machine translation: Parameter estimation. Computational linguistics, 19(2), pp. 263–311 (1993)
29. Pouliquen, B., Steinberger, R., Ignat, C.: Automatic Identification of Document Translations in Large Multilingual Document Collections. Proc. of the Int. Conf. Recent Advances in Natural Language Processing, pp. 401-408 (2003)
30. Pinto, D., Civera, J., Barrón-Cedeño, A., Juan, A., Rosso, P.: A Statistical Approach to Crosslingual Natural Language Tasks. Journal of Algorithms, 64(1), pp. 51-60 (2009)
31. Flores, E.,Barrón-Cedeño, A., Rosso, P., Moreno, L.: Towards the Detection of Cross-Language Source Code Reuse. Natural Language Processing and Information Systems, 31, LNCS, vol. 6716, pp. 250–253, Springer-Verlag (2011)
32. Simard, M., Foster, G., Isabelle, P.: Using cognates to align sentences in bilingual corpora. Proc. of the 1993 Conf. Centre for Advanced Studies on Collaborative research: distributed computing, IBM Press, 2, pp. 1071–1082 (1993)