Journal of Universal Computer Science, vol. 21, no. 13 (2015), 1708-1725
submitted: 4/5/15, accepted: 6/7/15, appeared: 28/12/15 © J.UCS

Cross-Language Source Code Re-Use Detection Using Latent Semantic Analysis

Enrique Flores
(Universitat Politècnica de Valencia, Spain
eflores@dsic.upv.es)

Alberto Barrón-Cedeño
(Qatar Computing Research Institute, HBKU, Qatar
albarron@{qf.org.qa|gmail.com})

Lidia Moreno, Paolo Rosso
(Universitat Politècnica de Valencia, Spain
{lmoreno,prosso}@dsic.upv.es)

Abstract: Nowadays, Internet is the main source to get information from blogs, encyclopedias, discussion forums, source code repositories, and more resources which are available just one click away. The temptation to re-use these materials is very high. Even source codes are easily available through a simple search on the Web. There is a need of detecting potential instances of source code re-use. Source code re-use detection has usually been approached comparing source codes in their compiled version. When dealing with cross-language source code re-use, traditional approaches can deal only with the programming languages supported by the compiler. We assume that a source code is a piece of text ,with its syntax and structure, so we aim at applying models for free text re-use detection to source code. In this paper we compare a Latent Semantic Analysis (LSA) approach with previously used text re-use detection models for measuring cross-language similarity in source code. The LSA-based approach shows slightly better results than the other models, being able to distinguish between re-used and related source codes with a high performance.

Key Words: Cross-Language Re-Use Detection, Source Code, Plagiarism, Latent Semantic Analysis
Category: D.2.13, F.3.2, F.3.3, H.3.1, H.3.3, H.3.4, I.2.5, I.7.0, L.0.0, L.3.0

1 Introduction

Text re-use[1] is an increasing problem in environments such as academia. Around 80% of high school students admit to have re-used contents at least once [Chapman and Lupton(2004)]. Source code re-use detection is important for academia and software companies. A student —or professional programmer— can retrieve a piece of source code from the Web and use it as her own without acknowledging the source. In an academic survey [Cosma and Joy(2008)], the

[1] We refer to re-use as hypernym of plagiarism, when we have evidence that a piece of content has been borrowed from a different document but there is no information about the author's consent.

majority of respondents agreed that, when re-use is permitted, students should adequately acknowledge the parts of the source code written by others; otherwise these actions could be interpreted as plagiarism. A programmer could even translate a piece of code into another programming language which, in the absence of proper credit or the consent of the original coder, might be considered as cross-language source code plagiarism. Moreover, detecting re-use is important for software companies in need of preserving the intellectual property of their programs.[2] Both scenarios are related to international programming competitions supported by software companies; e.g., the ACM-ICPC competition for college students[3] and Google's Code Jam[4]. Indeed, we have detected instances of source code re-use in Google Code Jam in the past [Flores et al.(2015)].

Manual retrieval of re-used documents from large collections is a difficult task; manual retrieval from the Web is unfeasible. Automatic re-use detection systems can easily retrieve re-used candidates on the fly. Over the last few years, re-use detection has caused a great interest promoting international tracks [Potthast et al.(2010)] for text re-use and for source code re-use.[5]

In this paper we compare the performance of a latent-semantic-based approach against other text re-use models that had been employed previously within the context of cross-language source code re-use detection [Flores et al.(2014a)]. We process source codes as simple texts and apply effective techniques of free text re-use detection [Clough(2000)]. Source code re-use has been explored in a cross-language perspective in just a few research works [Flores et al.(2014a), Arwin and Tahaghoghi(2006), Flores et al.(2011)]. We looked for instances of cross-language source code re-use on the most used programming languages nowadays: C, Java, and Python.[6]

The rest of the paper is organized as follows. In [Section 2] we overview different approaches to detect source code re-use. In [Section 3], we describe the used corpora, built out of the Rosettacode-source-code repository. [Section 4] describes the LSA-based model and the other 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 explored from different perspectives [Potthast et al.(2011), Stamatatos(2011), Brin et al.(1995)]. Monolingual analysis has been tackled with a wide range of approaches from n-grams [Barrón-Cedeño and

[2] `http://www.out-law.com/page-4613`
[3] `http://icpc.baylor.edu/`
[4] https://code.google.com/codejam
[5] `http://pan.webis.de/` and `http://www.dsic.upv.es/grupos/nle/soco/`
[6] `http://spectrum.ieee.org/computing/software/top-10-programming-languages`

Table 1: Concepts of comparability and parallelism in natural and programming language texts.

	Natural language text	Source code
Comparable documents	Documents that describe the same topic in different languages	Source codes that solve the same problem in different programming languages
Parallel documents	Documents that are translation of each other	Source codes that are considered (near-)translations

Rosso(2009), Stamatatos(2009)] or fingerprinting [Brin et al.(1995)] up to more semantic approaches [Soleman(2014)].

Cross-language text analysis has been mostly tackled with four different models: *(i)* dictionary-based [Gupta et al.(2012)], built on top of vector space models; *(ii)* syntax-based, these models are able to capture syntactical similarities and similar vocabulary between languages [Mcnamee and Mayfield(2004)]; *(iii)* comparable corpora [Potthast et al.(2008)]; and *(iv)* parallel corpora [Dumais et al.(1997), Littman et al.(1998), Ceska(2009), Barrón-Cedeño et al.(2008)]. The last two kinds of models require to be trained on aligned corpora in the two languages. Comparable approaches need documents on approximately the same topic in the two languages. Parallel approaches need documents that are translations of each other in order to infer a relation between the words in the two languages. An overview can be found in [Potthast et al.(2011)].

Regarding natural language text, comparable documents are those covering the same topic and written in different languages [Potthast et al.(2008)]. Parallel documents are two documents which are a translation of each other [Steinberger et al.(2006)]. Similar definitions are possible for programming languages. We consider that two codes are comparable if they solve the same problem but have been independently written in different programming languages. Comparability does not imply re-use. We consider two codes are parallel if they are (near-)translations of each other. That is, parallelism does imply re-use. [Tab. 1] compares the concepts of comparable and parallel documents in both natural and programming language texts. This conceptual parallelism opens the door to borrow models originally intended to uncover re-use in natural language text to uncover re-use in source code.

Most of the work carried out in source code re-use detection intended to approach monolingual. Automatic source code re-use detection has been approached from two main perspectives: attribute counting and structure [Clough(2000)]. Attribute counting uses different characteristics of the code, such as the number of identifiers [Halstead(1972)], the average nesting depth [McCabe(1976)], or the number of calls to a given function [Selby(1989)].

Structure-based approaches have been more widely explored because they can detect cases of re-use even after the source code has been altered [Whale(1990)]. In the structure-based approaches, the detection process is carried out in three steps: *(i)* pre-process the code by removing comments, spaces, and punctuation marks; *(ii)* represent the source code using its structure; and *(iii)* compare the structural representations of two codes and estimate their similarity. In order to represent a program structure, in [Whale(1990)], the author proposed to use the bifurcation statements of the source code as the comparison unit and the kind of bifurcations to determine their similarity. This work inspired other approaches —such as YAP3 [Wise(1992)] and JPlag [Prechelt et al.(2002)]—, which also use efficient string-matching algorithms to find similar code fragments.

Other proposals use the output of a compiler to represent the structure of the source code. In [Chilowicz et al.(2009)], the authors compare the syntax tree generated during the compiler 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 dependencies graph [Krinke(2001)]. This approach is powerful against changes in the structure of the source code (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 the search of isomorphic sub-graphs is NP-hard. These proposals are compiler-dependent. As a result, they can be used only on codes developed within programming frameworks, such as the GNU Compiler Collection (GCC) or Microsoft Visual Studio[7].

An approach that requires less resources (i.e. it does not need a compiler) is based on the Winnowing algorithm [Marinescu et al.(2013)]. Initially, it selects the reserved words, delimiters, and operators of the programming languages. The resulting fragments of source codes are represented with hashes. Although this model is efficient, it is sensitive to modifications: a change of a character in a string changes the hash value completely, causing possible re-use cases to go undetected.

Another effective approach that does not require any information of the programming language is based on Latent Semantic Analysis (LSA) [Cosma and Joy(2012)]. The pre-processing consists of removing tokens solely composed of numeric characters, syntactical tokens (e.g., semicolons, colons), tokens occurring in less than two files, and one-character tokens. Then, LSA is applied using term-frequency for weighting and the cosine measure to compute the similarity, after a reduction of the representation vectors to 30 dimensions. In [Cosma and Joy(2012)], the authors compared this model with the well-known JPlag string-matching model. They also measured the correlation between the LSA results

[7] https://gcc.gnu.org/ and https://www.visualstudio.com/

and human judgements. The LSA-based model achieved slightly better performance than JPlag in different scenarios and it showed a high correlation with human judgement.

To the best of our knowledge, only [Arwin and Tahaghoghi(2006)] has tackled the problem of cross-language source code re-use detection. This is an interesting compiler-dependent approach in which programs in different languages are compiled to generate a common intermediate code representation (e.g., with the GCC), which is directly comparable. Obviously, this approach is strongly compiler-dependent and could be limited to short amounts of programming languages.

3 Corpora

We extracted our corpus from the Rosettacode repository. Rosettacode[8] is a website that presents solutions to the same task in as many different programming languages as possible. In a snapshot of Feb. 27, 2012, there were 600 solved tasks 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 build a comparable corpus.

Our procedure to construct parallel instances from the original Rosetta codes is as follows. We translate all the comparable source codes written in C into Java using *C++ to Java Converter*[9]. The translated Java code was then translated into Python with *java2python*[10]. After translation, comments introduced by the translators were removed. Note that *java2python* generates a syntactically (almost) equal to the original Java source code, whereas *C++ to Java Converter* is able to rephrase and refactor source code if necessary. This implies differences in code lengths between translations. [11]

[Tab. 2] shows the amount of comparable and parallel source code pairs between the different programming languages. We divide the documents into training and test collections: 70% is used to train the respective retrieval model and 30% for test.

4 Models

In this paper we compare six models in the problem of cross-language source code re-use detection. Three models require external resources: Cross-Language

[8] `http://rosettacode.org/`

[9] `http://www.tangiblesoftwaresolutions.com/Product\_Details/CPlusPlus\_to\_Java\_Converter\_Details.html`

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

[11] The resulting corpus is freely available for research purposes at `http://users.dsic.upv.es/grupos/nle/?file=kop4.php`.

Table 2: Number of comparable and parallel source code pairs per each pair of programming languages in the corpus.

	C-Java	Java-Python	C-Python	Size (MB)
Comparable	959	1588	1408	3.54
Parallel	335	335	335	2.12

Latent Semantic Analysis (CL-LSA), Cross-Language Explicit Semantic Analysis (CL-ESA) and Cross-Language Alignment-based Similarity Analysis (CL-ASA). Three other models require no resources at all: Cross-Language Character 3-grams (CL-C3G), Pseudo-cognates (COG), and Word Count Ratio (WCR). All the models but CL-LSA have been applied to this task before [Flores et al.(2014a)]. CL-LSA is reported to achieve a high retrieval quality [Dumais et al.(1997)]. CL-ESA has shown good performance when looking for comparable documents and CL-ASA has shown accurate results when looking for parallel documents scenario [Potthast et al.(2011)]. CL-C3G, COG and WCR, have shown to be worth considering to assess similarities within comparable corpora [Barrón-Cedeño et al.(2014)]. Following, we present our adaptations of these models to the source code scenario.

4.1 Cross-Language Latent Semantic Analysis (CL-LSA)

Most IR models rely on exact matches between query and document words. As a result, the lack of common words between query and document causes these models to fail. The main reason is that standard models (e.g., Boolean model) process each word as an independent dimension when in fact they are not independent. LSA allows to model the relations between words of the same document and improves the retrieval process [Deerwester et al.(1990)]. Dumais [Dumais(1995)] describes experiments on TREC benchmarks giving evidence that, at least in some cases, LSA can produce better precision and recall than standard vector-space retrieval.

Following the principle that similar words appear in similar contexts, LSA examines the similarity between the “contexts” in which a word appears and creates a new vector space with less dimensions. LSA uses Singular Value Decomposition (SVD) to discover the most important relations between terms from a collection of documents. These automatically-estimated relations are specific to the domain of the collection.

The SVD technique is similar to the eigenvalue decomposition of a matrix (*eigenvector*) and to factorial analysis [Cullum and Willoughby(2002)]. It starts constructing a term matrix from the collection as in the Boolean or vector space model [Baeza-Yates and Ribeiro-Neto(2011)]. This matrix is decomposed in a set of k orthogonal factors calculated by lineal approximation. This approximation "discovers" the latent structure in the matrix from the term frequency of each document. As a result, SVD returns a set of vectors that represent the situation of each term and document in a reduced vector space of k dimensions.

LSA represents the terms as continuous values on each orthogonal dimension. When two terms appear in two similar contexts, these terms will have similar vectors in the LSA-reduced matrix. The model solves partially the weakness of assuming independence between terms. Further mathematical descriptions and examples of the LSA and SVD principles are included in [Deerwester et al.(1990)].

LSA can be adapted to cross-language IR by concatenating each pair of parallel documents for training [Landauer and Littman(1990)]. This set is analyzed using LSA and a new reduced k-dimensional vector space is computed. As the training documents contain terms in both languages, the relations are calculated between terms in both languages. Using these cross-language relations, a translation of the query is not necessary; terms in both languages will have a similar representation in the reduced k-dimensional vector space. This allows language independent searches on the basis of previously-computed relations between terms in different languages.

LSA has been applied in monolingual source code re-use detection already [Cosma and Joy(2012)]. The authors consider the source code as a piece of text in order to create a term representation. Their overall results suggest that integrating LSA with the tools Sherlock and JPlag improves the overall detection performance.

To the best of our knowledge, this is the first attempt to apply LSA for cross-language source code re-use. As in texts, the model can be adapted by concatenating parallel source codes for training. Equivalent terms between programming languages will achieve a high similarity value.

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

This model was originally proposed to compute the similarity between texts across languages using a comparable corpus (e.g., from Wikipedia). In the cross-language source code re-use detection scenario, a source code d (d') written in the programming language L (L') is compared against each source code from a comparable collection $\{C, C'\}$, written in language L (L'). We consider that two programs $x_i \in C, x'_i \in C'$ are comparable if they solve the same problem. Two source codes in different languages can be compared against the comparable collection to generate the corresponding representation vectors:

$$\overrightarrow{d} = \{sim(d, x) \forall x \in C / C \in L\} \quad (1)$$

Table 3: Estimated length factors for each language pair measured in tokens. A value of $\mu > 1$ implies that, on average, $len(d) < len(d')$ for d and its translation d' in terms of the number of reserved words.

Parameter	C→Java	Java→C	Java→Python	Python→Java	C→Python	Python→C
μ	0.925	1.463	1.003	1.008	0.989	1.564
σ	0.505	3.089	0.077	0.083	0.799	2.595

$$\overrightarrow{d'} = \{sim(d', x') \forall x' \in C' / C' \in L'\} \tag{2}$$

We compute cosine similarities over frequency-weighted character 3-grams to produce the vector representations. Later on, the cosine similarity between vectors $\overrightarrow{d}$ and $\overrightarrow{d'}$ is computed to indirectly estimate the similarity between d and d'. Although the "semantic" component of the model may not be too obvious when CL-ESA is applied to detect cross-language source code re-use, the model allows to detect instances in which two source codes are written in semantically similar ways (e.g., *for* and *while* statements).

4.3 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 [Brown et al.(1993)]. The translation model represents the probability that the source string is the translation of the target string, and the language model is the probability of seeing that target language string. In CL-ASA, the language model is substituted by a length model, which determines the likelihood of a code being a translation of one another according to its expected length. The length model, originally proposed by [Pouliquen et al.(2003)], is defined as:

$$lf(d, d') = e^{-0.5\left(\frac{len(d)/len(d') - \mu}{\sigma}\right)} \tag{3}$$

where μ and σ are the mean and standard deviation of the lengths ratios between translations of source codes from language L into L' and the *len* function returns the number of reserved words in the source code. [Tab. 3] shows the values of μ and σ that we estimated for the considered language pairs over the parallel training collection described in [Section 3].

We use the translation model to determine if the tokens in code d are valid translations of the tokens in code d'. This model requires a statistical bilingual dictionary, usually estimated from a sentence-aligned parallel corpus. For source code, we consider whole programs as parallel units, and limit the vocabulary to reserved words, operators, and identifiers. Identifiers are substituted by the generic identifier "*id*". To estimate the translation probabilities, we use the IBM 1 alignment model [Brown et al.(1993)].

In [Pinto et al.(2009)] we proposed an adaptation to the translation model, originally intended to handle sentences, to estimate the similarity between entire documents. The adapted translation model is defined as:

$$w(d|d') = \sum_{x \in d} \sum_{y \in d'} p(x, y) \ , \tag{4}$$

where $p(x, y)$ represents the probability of x of being a translation of y 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.

The similarity $\varphi(d, d')$ between d and d' results of the combination of this adapted translation model and the length model:

$$\varphi(d, d') = lf(d, d') \cdot w(d|d') \ , \tag{5}$$

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

We explored four dictionaries for the translation model: *(i)* a statistical dictionary built from a parallel corpus using IBM model 1; *(ii)* the same dictionary as built in *(i)* , but considering only the most likely translations of each entry with a probability mass of 20%; *(iii)* same as *(ii)* , with a probability mass of 40%; and *(iv)* a dictionary generated from conversion table between the reserved words of the programing languages, extracted from Wikipedia[12]. We considered the model that uses the 40% most likely of being translation from the dictionary because it achieved a slightly better performance than the other CL-ASA variants.

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

Character n-grams have been used to calculate text similarity across languages that regularly preserve the morphology of the words (e.g. using inflections or derivations) [Goldsmith et al.(2001)]. Numerous programming languages with morphological similarities exist; e.g., C and C++ or Java and C++. Similarity models based on morphology rely on these similarities and the inherited vocabulary from other languages. This similarity can be well estimated using a Vector Space Model on short terms, such as character n-grams. Character n-grams have shown good accuracy for monolingual and cross-language information retrieval [Mcnamee and Mayfield(2004)]. As we showed previously, they can achieve good results when applied to cross-language source code re-use detection [Flores et al.(2011)].

[12] `http://en.wikipedia.org/wiki/Comparison\_of\_programming\_languages\_\%28basic\_instructions\%29` - Accessed on 14th March 2014

The pre-processing in this model consists of removing line feeds, tabs, and spaces; characters are 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 then estimated using the cosine measure.

4.5 Pseudo-Cognateness (CL-COG)

Cognates are defined as pairs of tokens of different languages which, usually share some phonological or morphological properties [Voga and Grainger(2007)] As a result, they are likely to be used as mutual translations [Simard et al.(1993)]. Our cognate candidates include those tokens with at least one digit and those tokens composed by three letters or more. Examples of pseudo-cognates are *for–foreach* in languages C and C# or *Integer–int* in Visual Basic and C++ respectively. We use the term pseudo-cognates because in source codes we are able to capture as possible cognate any variable name that is not necessarily the same term derived or inflected as in natural languages (e.g. $variable$ and $variable\_modified$). Using pseudo-cognates, we determine how related two pieces of source code are.

The pre-processing in this case consists of case-folding. Then, the cognate candidates of each source code are extracted. The similarity is calculated as:

$$\varphi(d, d') = \frac{c}{(m+n)/2} \quad , \tag{6}$$

where parameters m and n are the number of cognate candidates in each source code and c is the number of matches between the two lists of cognate candidates so as to obtain the maximum number of pairs without using the same token in different matches.

4.6 Word Count Ratio (WCR)

In the WCR model, tokens are extracted from the source codes and delimiters (e.g. semicolons, brackets, etc.) are discarded. Afterwards, each source code is represented by its length in terms of tokens. The similarity is computed as the length ratio between the shorter and the longer source code in number of tokens:

$$\varphi(d, d') = \frac{min(len(d), len(d'))}{max(len(d), len(d'))} \quad . \tag{7}$$

WCR has shown a high correlation with human judgements measuring cross-lingual similarity in texts [Barrón-Cedeño et al.(2014)], regardless its simplicity and non-use of external resources.

Table 4: Area Under the Precision-Recall curve for each model and programming language pair.

Lang. pair	CL-LSA	CL-C3G	CL-ESA	CL-ASA	CL-COG	WCR
C–Java	**0.768**	**0.770**	0.603	0.470	0.708	0.690
Java–Python	**0.969**	0.797	0.411	0.399	0.523	0.459
C–Python	**0.365**	0.296	0.238	0.213	0.294	0.322

5 Experiments

In our experiments we compare a LSA-based model against other IR models in the task of detecting cross-language source code re-use. In our previous work, we compared the performance of those IR models in the task of cross-language source code retrieval; looking for parallel and comparable programs [Flores et al.(2014a)]. Now we consider a more realistic scenario. In the Rosettacode.org repository, multiple contributors post alternative solutions to the same algorithmic problem. They could be derived from implementations in another programming language or be coded from scratch. Re-used programs across languages could have a similar structure or share some methods or chunks. The entire program could be simply translated from one language into one another. We created two partitions out of the implementations of a solution in different programming languages available in Rosetta: re-used instances are composed of parallel source code pairs; non-re-used instances include comparable source code pairs.

5.1 Performance of the Models

The models are evaluated on the basis of Precision-Recall curves and area under the curve (AUC) in three programming language pairs: C–Java, Java–Python and C–Python. [Fig. 1] shows the resulting Precision-Recall curves whereas [Tab. 4] shows the values of AUC. CL-LSA shows a similar performance than CL-C3G when facing the C–Java pair. The resulting AUC values are roughly the same. In the other pairs, the CL-LSA curve is over the others most of the times. LSA-based model outperforms CL-C3G as well facing monolingual source code re-use [Flores et al.(2014b)]

As we observed before the rest of models perform similarly when retrieving parallel and comparable documents in isolation [Flores et al.(2014a)]. In this scenario CL-C3G and CL-COG achieve a good performance compared to WCR, CL-ESA and CL-ASA's relatively lower results.

Regarding the different language pairs, C–Java and Java–Python allow for better performance. CL-LSA shows nearly-perfect results in the Java–Python

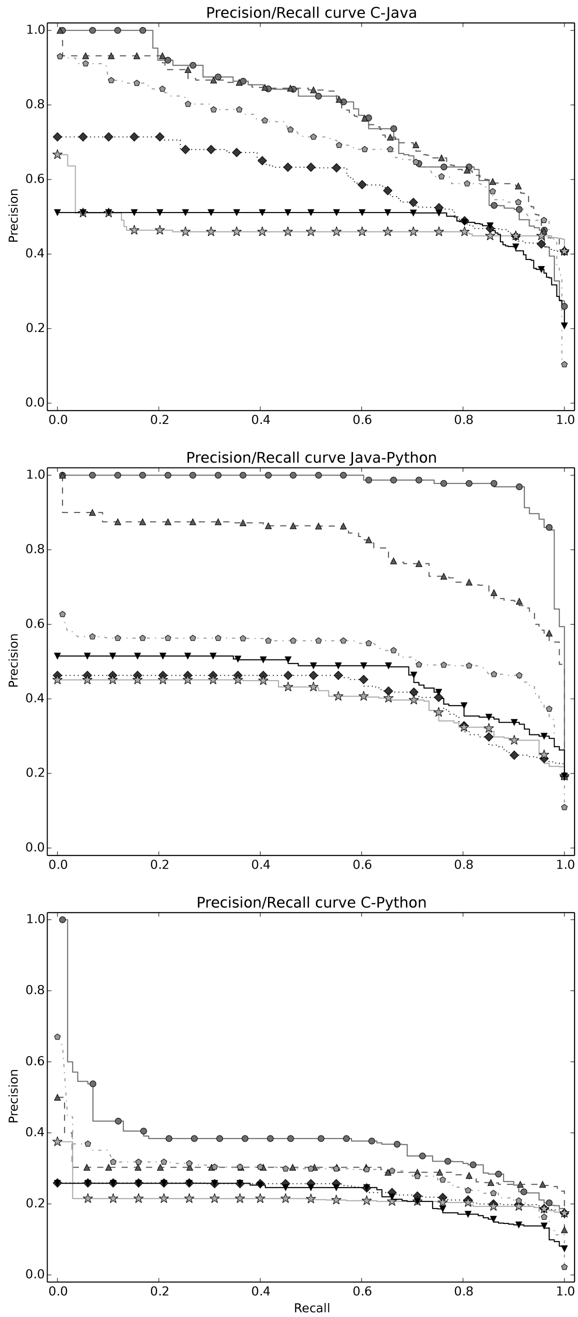

Figure 1: Precision-recall curves for each model and programming language pair.

pair. The reason could be in the *java2python* translator, which produces almost syntactically-identical translations. Another evidence of this phenomenon is the length factor parameters, reported in [Section 4.3]: μ is close to 1, and the standard deviation is small. That is, the translations have an almost equal number of tokens than the original source codes. The *C++ to Java Converter* needs to transform the source code into refactored source code. Even if the translator did not recognize a certain function call, it would generate a new void function in order to be completed by the programmer. In the C–Python scenario, this parallel source codes are the result of two translations while the source codes of the other programming language pairs only needed one translation process.

5.2 Similarity threshold

In this subsection, our aim is to set a similarity threshold for the best-performing models: CL-LSA and CL-C3G (as shown in [Fig. 1]). The similarity threshold is the lowest similarity value from which a source code pair can be considered an instance of re-use. We estimate the similarity threshold out of the parallel and comparable similarity distributions considering the 66% of the test partition. We represent the similarity in steps of 0.05 and the amount of source code pairs has been normalized. [Fig. 2] presents the distribution of re-used and non-re-used source codes in terms of the similarity for CL-LSA, CL-C3G and CL-ASA models. CL-ASA is added for comparison purposes.

Both CL-LSA and CL-C3G show non-completely separated distributions, whereas CL-ASA shows highly overlapped distributions. As a result, with the former models it is possible to discriminate whether a source code has been re-used with certain confidence. We set the discriminative threshold as the point where the polynomial trend lines of the distributions cross. The overlapped distributions of CL-ASA make it impossible to set a threshold.

The performance of the models after setting the threshold is measured using Precision, Recall and F_1 using the remaining 33% of the test partition. [Tab. 5] shows the obtained results. In the C–Java scenario both models show similar performance but in the other scenarios CL-LSA shows a higher F_1 measure as it occurred in the Precision-Recall curves. In general, both models achieve good results with high values of precision, recall and F_1. The models show the same trend than in the Precision–Recall curves, achieving lower performance in the pair C–Python.

6 Conclusions and Future Work

In this paper we approached the problem of cross-language source code re-use detection. In our experiments, we considered a realistic scenario taking advantage of a collection of source code pairs from `Rosettacode.org`: parallel pro-

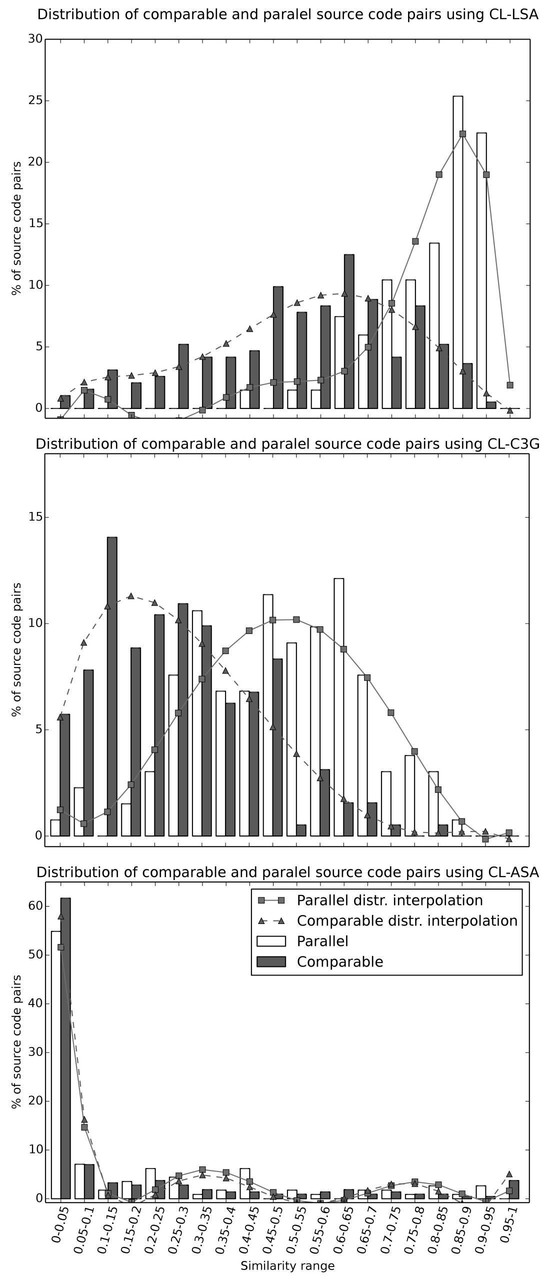

Figure 2: Similarity distributions for CL-LSA, CL-C3G and CL-ASA (added for comparison) using C–Java test by ranges of 0.05. A polynomial trend line of degree 6 is plotted for each distribution.

Table 5: Evaluation of the CL-LSA and CL-C3G models after setting the appropriate decision threshold from the parallel and comparable distributions.

Model	Pair	Threshold	Precision	Recall	F_1
CL-LSA	C–Java	0.750	0.674	0.853	**0.753**
	Java–Python	0.770	0.756	1.000	**0.861**
	C–Python	0.550	0.304	0.824	**0.444**
CL-C3G	C–Java	0.370	0.630	0.773	0.694
	Java–Python	0.410	0.526	0.882	0.659
	C–Python	0.270	0.280	0.618	0.385

gram pairs were considered as positive instances of re-use, whereas comparable pairs were considered as originals. We compared a latent semantic analysis-based model against other well-known information retrieval models originally intended to handle plain text, including cross-language character n-grams and cross-language explicit semantic analysis.

The model based on latent semantic analysis performed the best in this task, showing more accurate than the other models when classifying instances in three programming language pairs: C–Java, C–Python, and Java–Python. The second best model, based on character n-grams, was competitive when facing the C–Java pair only. A deeper analysis showed that, contrary to the other models, the similarity distributions obtained with both latent semantic analysis and character n-grams overlap just slightly, causing the definition of good discriminative thresholds possible.

As future work, we plan to extrapolate these estimated cross-language thresholds using LSA-based and character n-gram based models in another cross-language source code re-use scenario.

Acknowledgements

This work was partially supported 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.

References

[Arwin and Tahaghoghi(2006)] Arwin, C., Tahaghoghi, S.: "Plagiarism detection across programming languages"; Proceedings of 29th Australasian Computer Science Conference, 48, (2006), 277–286.

[Baeza-Yates and Ribeiro-Neto(2011)] Baeza-Yates, R. A., Ribeiro-Neto, B. A.: "Modern Information Retrieval - the concepts and technology behind search"; Second edition, Pearson Education Ltd., Harlow, England, (2011) [Barrón-Cedeño et al.(2014)] Barrón-Cedeño, A., Paramita, M. L., Clough, P., Rosso, P.: "A comparison of approaches for measuring cross-lingual similarity of Wikipedia articles"; Proceedings European Conference Information Retrieval, LNCS(8416), Springer, (2014), 424–429.

[Barrón-Cedeño and Rosso(2009)] Barrón-Cedeño, A., Rosso, P.: "On automatic plagiarism detection based on n-grams comparison"; Proceedings of the 31st European Conference on Information Retrieval, LNCS(5478), Springer-Verlag, (2009), 696–700.

[Barrón-Cedeño et al.(2008)] Barrón-Cedeño, A., Rosso, P., Pinto, D., Juan, A.: "On cross-lingual plagiarism analysis using a statistical model"; ECAI 2008 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN-08), (2008), 9–13.

[Brin et al.(1995)] 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, (1995), 398–409.

[Brown et al.(1993)] Brown, P., Pietra, V., Pietra, S., Mercer, R.: "The mathematics of statistical machine translation: Parameter estimation"; Computational linguistics, 19, 2, (1993), 263–311.

[Ceska(2009)] Ceska, Z.: "Automatic plagiarism detection based on latent semantic analysis"; Ph.D. thesis, Faculty Apploed Science, University of West Bohemia, Czech Republic (2009).

[Chapman and Lupton(2004)] Chapman, K., 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, (2004), 425–435.

[Chilowicz et al.(2009)] Chilowicz, M., Duris, E., Roussel, G.: "Syntax tree fingerprinting for source code similarity detection"; IEEE 17th International Conference on Program Comprehension, (2009), 243–247.

[Clough(2000)] Clough, P.: "Plagiarism in natural and programming languages: An overview of current tools and technologies"; Technical report, Department of Computer Science, University of Sheffield (2000).

[Cosma and Joy(2012)] Cosma, G., Joy, M.: "An approach to source-code plagiarism detection and investigation using latent semantic analysis"; IEEE Transactions on Computers, 61, 3, (2012), 379–394.

[Cosma and Joy(2008)] Cosma, G., Joy, M.: "Towards a definition of source-code plagiarism"; IEEE Transactions on Education, 51, 2, (2008), 195–200.

[Cullum and Willoughby(2002)] Cullum, J., Willoughby, R.: "Lanczos Algorithms for Large Symmetric Eigenvalue Computations"; Society for Industrial and Applied Mathematics, (2002).

[Deerwester et al.(1990)] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: "Indexing by latent semantic analysis"; Journal of the American Society for Information Science, 41, 6, (1990), 391–407.

[Dumais et al.(1997)] Dumais, S., Letsche, T., Littman, M., Landauer, T.: "Automatic cross-language retrieval using latent semantic indexing"; AAAI-97 Spring Symposium Series: Cross-language text and speech retrieval, (1997), 18–24.

[Dumais(1995)] Dumais, S. T.: "Latent semantic indexing (LSI): Trec-3 Report"; Proceedings of TREC, (1995), 219–230.

[Flores et al.(2015)] Flores, E., Barrón-Cedeño, A., Moreno, L., Rosso, P.: "Uncovering source code reuse in large-scale academic environments"; Computer Applications in Engineering Education, 23, 3, (2015), 383–390.

[Flores et al.(2014a)] Flores, E., Barrón-Cedeño, A., Moreno, L., Rosso, P.: "Cross-language source code re-use detection"; Proceedings of the 3rd Spanish Conference on Information Retrieval, (2014), 145–156.

[Flores et al.(2014b)] Flores, E., Ibarra-Romero, M., Moreno, L., Sidorov, G., Rosso, P.: "Modelos de recuperación de información basados en n-gramas aplicados a la reutilización de cdigo fuente"; Proceedings of the 3rd Spanish Conference on Information Retrieval, (2014), 185–188.

[Flores et al.(2011)] Flores, E., Barrón-Cedeño, A., Rosso, P., Moreno, L.: "Towards the detection of cross-language source code reuse"; Natural Language Processing and Information Systems, LNCS(6716), Springer Berlin Heidelberg, (2011), 250–253.

[Goldsmith et al.(2001)] Goldsmith, J. A., Higgins, D., Soglasnova, S.: "Automatic language-specific stemming in information retrieval"; Revised Papers from the Workshop of CLEF on Cross-Language Information Retrieval and Evaluation, Springer, (2001), 273–283.

[Gupta et al.(2012)] Gupta, P., Barrón-Cedeño, A., Rosso, P.: "Cross-language high similarity search using a conceptual thesaurus"; Proceedings of the 3rd International Conference of CLEF Initiative on Information Access Evaluation meets Multilinguality, Multimodality, and Visual Analytics, LNCS(7488), Springer-Verlag, (2012), 67–75.

[Halstead(1972)] Halstead, M.: "Naturals laws controlling algorithm structure?"; SIGPLAN Notices, 7, 2, (1972).

[Krinke(2001)] Krinke, J.: "Identifying similar code with program dependence graphs"; IEEE Proceedings of the 8th Conference on Reverse Engineering, (2001), 301–309.

[Landauer and Littman(1990)] Landauer, T., Littman, M.: "Fully automatic cross-language document retrieval using latent semantic indexing"; Proceedings of the 6th Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research, (1990), 31–38.

[Littman et al.(1998)] Littman, M., Dumais, S., Landauer, T.: "Automatic cross-language information retrieval using latent semantic indexing"; Cross-language information retrieval, Springer US., (1998), 51–62.

[Marinescu et al.(2013)] Marinescu, D., Baicoianu, A., Dimitriu, S.: "A plagiarism detection system in computer source code"; International Journal Computer Science Research and Application, 3, 1, (2013), 22–30.

[McCabe(1976)] McCabe, T.: "A complexity measure"; IEEE Transactions on Software Engineering, 4, 2, (1976), 308–320.

[Mcnamee and Mayfield(2004)] Mcnamee, P., Mayfield, J.: "Character n-gram tokenization for european language text retrieval"; Information Retrieval, 7, 1-2, (2004), 73–97.

[Pinto et al.(2009)] Pinto, D., Civera, J., Barrón-Cedeño, A., Juan, A., Rosso, P.: "A statistical approach to crosslingual natural language tasks"; Journal of Algorithms, 64, 1, (2009), 51–60.

[Potthast et al.(2011)] Potthast, M., Barrón-Cedeño, A., Stein, B., Rosso, P.: "Cross-language plagiarism detection."; Language Resources and Evaluation, Special Issue on Plagiarism and Authorship Analysis, 45, 1, (2011), 45–62.

[Potthast et al.(2010)] Potthast, M., Barrón-Cedeño, A., Stein, B., Rosso, P.: "An evaluation framework for plagiarism detection."; ACL Proceedings of the 23rd International Conference on Computational Linguistics, (2010), 997–1005.

[Potthast et al.(2008)] Potthast, M., Stein, B., Anderka, M.: "A Wikipedia-based multilingual retrieval model"; Proceedings of the 30th European Conference on IR Research, LNCS(4956), Springer Berlin Heidelberg, (2008), 522–530.

[Pouliquen et al.(2003)] Pouliquen, B., Steinberger, R., Ignat, C.: "Automatic identification of document translations in large multilingual document collections"; Proceedings of the International Conference Recent Advances in Natural Language Processing, (2003), 401–408.

[Prechelt et al.(2002)] Prechelt, L., Malpohl, G., Philippsen, M.: "Finding plagiarisms among a set of programs with JPlag"; Journal of Universal Computer Science, 8, 11, (2002), 1016–1038.

[Selby(1989)] Selby, R.: "Quantitative studies of software reuse"; ACM Software Reusability, 2, (1989), 213–233.

[Simard et al.(1993)] Simard, M., Foster, G., Isabelle, P.: "Using cognates to align sentences in bilingual corpora"; Proceedings of the Conference Centre for Advanced Studies on Collaborative research: Distributed Computing, IBM Press, 2,, (1993) 1071–1082.

[Soleman(2014)] Soleman, S.: "Experiments on the indonesian plagiarism detection using latent semantic analysis"; Proceedings of the 2nd International Conference on Information and Communication Technology, (2014), 413–418.

[Stamatatos(2011)] Stamatatos, E.: "Plagiarism detection using stopword ngrams"; Journal of the American Society for Information Science and Technology, 62, 12, (2011), 2512–2527.

[Stamatatos(2009)] Stamatatos, E.: "Intrinsic plagiarism detection using character n-gram profiles"; Proceedings of the 3rd International Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse, (2009), 38–46.

[Steinberger et al.(2006)] 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"; Proceedings of the 5th International Conference on Language Resources and Evaluation, (2006).

[Voga and Grainger(2007)] Voga, M., Grainger, J.: "Cognate status and cross-script translation priming"; Memory & Cognition, 35, 5, (2007), 938–952.

[Whale(1990)] Whale, G.: "Software metrics and plagiarism detection"; Journal of Systems and Software, 13, 2, (1990), 131–138.

[Wise(1992)] Wise, M.: "Detection of similarities in student programs: Yaping may be preferable to plagueing"; Proceedings of the 23th SIGCSE Technical Symposium, (1992), 268–271.