Evaluating the Performance of LSA for Source-code Plagiarism Detection

Georgina Cosma
Department of Business Computing, PA College, Larnaca, CY-7560 Cyprus
E-mail: g.cosma@faculty.pacollege.ac.cy

Mike Joy
Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK
E-mail: M.S.Joy@warwick.ac.uk

Keywords: LSA, source-code similarity detection, parameter tuning

Received: October 25, 2012

Latent Semantic Analysis (LSA) is an intelligent information retrieval technique that uses mathematical algorithms for analyzing large corpora of text and revealing the underlying semantic information of documents. LSA is a highly parameterized statistical method, and its effectiveness is driven by the setting of its parameters which are adjusted based on the task to which it is applied. This paper discusses and evaluates the importance of parameterization for LSA based similarity detection of source-code documents, and the applicability of LSA as a technique for source-code plagiarism detection when its parameters are appropriately tuned. The parameters involve preprocessing techniques, weighting approaches; and parameter tweaking inherent to LSA processing – in particular, the choice of dimensions for the step of reducing the original post-SVD matrix. The experiments revealed that the best retrieval performance is obtained after removal of in-code comments (Java comment blocks) and applying a combined weighting scheme based on term frequencies, normalized term frequencies, and a cosine-based document normalization. Furthermore, the use of similarity thresholds (instead of mere rankings) requires the use of a higher number of dimensions.

Povzetek: Prispevek analizira metodo LSA posebej glede plagiarizma izvirne kode.

1 Introduction

Latent Semantic Analysis (LSA) is an intelligent information retrieval technique that uses mathematical algorithms for analyzing large corpora of text and revealing the underlying semantic information of documents [10, 11]. Previous researchers have reported that LSA is suitable for textual information retrieval and is typically used for indexing large text collections and retrieving documents based on user queries. In the context of text retrieval, LSA has been applied to a variety of tasks including indexing and information filtering [12], essay grading [23, 38, 13, 43, 19, 18] cross-language information retrieval [44], detecting plagiarism in natural language texts [7], and source-code clustering and categorization [20, 25, 22, 26, 27, 28]. LSA has been applied to source-code with the aim of categorizing software repositories in order to promote software reuse [27, 28, 24] and much work has been done in the area of applying LSA to software components. Some of the LSA based tools developed include MUDABlue [20] for software categorization, Softwarenaut [25] for exploring parts of a software system using hierarchical clustering, and Hapax [22] which clusters software components based on the semantic similarity between their software entities (entire systems, classes and methods). Although LSA has been applied to source-code related tasks such as reuse and categorization of source-code artifacts, there appears to be a lack of literature investigating the behavior of parameters driving the effectiveness of LSA for tasks involving source-code corpora. The current literature also lacks an evaluation of LSA and its applicability to detecting source-code plagiarism [31, 32].

2 A Review of Latent Semantic Analysis

Latent Semantic Analysis uses statistics and linear algebra to reveal the underlying "latent" semantic meaning of documents [5]. *Latent Semantic Indexing* (LSI) is a special case of LSA, and the term LSI is used for tasks concerning the indexing or retrieval of information, whereas the term LSA is used for tasks concerned with everything else, such as automatic essay grading and text summarization.

The first step prior to applying LSA involves preprocessing the documents in the corpus in order to efficiently represent the corpus as a term-by-document matrix. Document pre-processing operations include the following [2].

– *Tokenization.* This involves identifying the spaces in

the text as word separators, and considering digits, hyphens, punctuation marks, and the case of letters.

- *Stopword elimination.* This is the elimination of words with a high frequency in the document corpus, and involves removing prepositions, conjunctions and common words that could be considered as useless for purposes of retrieval, e.g. words such as *the*, *and*, and *but*, found in the English language. In source-code this involves removing programming language reserved words (i.e. keywords).

- *Stemming of words.* This involves transforming variants of words with the same root into a common concept. A stem is the portion of the remaining word after removing its affixes (suffixes and prefixes). An example of a stem is the word *eliminat* which is the prefix of the variants *eliminated*, *eliminating*, *elimination*, and *eliminations*.

After pre-processing is performed, the corpus of documents is transformed into a $m \times n$ matrix $A = [a_{ij}]$, in which each row m represents a term vector, each column n represents a document vector, and each cell a_{ij} of the matrix A contains the frequency at which a term *i* appears in document *j*. Thus, the rows of matrix A represent the term vectors, and the columns of matrix A represent the document vectors.

Term weighting is then applied to matrix A. The purpose of term-weighting is to adjust the frequency values of terms using *local* and *global* weights in order to improve retrieval performance. *Local weights* determine the value of a term in a particular document, and *global weights* determine the value of a term in the entire document collection. Various local and global weighting schemes exist [4] and these are applied to the term-by-document matrix to give high weights to important terms, i.e. those that occur distinctively across documents, and low weights to terms that appear frequently in the document collection.

Document length normalization [41] adjusts the term values depending on the length of each document in the collection. The value of a term in a document is $l_{i,j} \times g_i \times n_j$, where $l_{i,j}$ is the local weighting for term i in document j, g_i is the global weighting for term i, and n_j is the document-length normalization factor [4]. Long documents have a larger number of terms and term frequencies than short documents and this increases the number of term matches between a query and a long document, thus increasing the retrieval chances of long documents over small ones. Literature claims that the *cosine document length normalization* can improve retrieval performance [41, 40].

Tables 1, 2, and 3 contain some of the most commonly used term-weighting formulas [4]. Symbol f_{ij} defines the number of times (term-frequency) term *i* appears in document *j*; let

$$b(f_{ij}) = \begin{cases} 1, & \text{if } f_{ij} > 0, \\ 0, & \text{if } f_{ij} = 0, \end{cases}$$

$$p_{ij} = \frac{f_{ij}}{\sum_j f_{ij}}$$

Once term-weighting is applied, the matrix is then submitted for Singular Value Decomposition (SVD) to derive the latent semantic structure model. Singular Value Decomposition decomposes matrix A into the product of three other matrices: an $m \times r$ term-by-dimension matrix, *U*, an $r \times r$ singular values matrix, Σ, and an $n \times r$ document by dimension matrix, V. The rank *r* of matrix A is the number of nonzero diagonal elements of matrix Σ. SVD can provide a rank-k approximation to matrix A, where *k* represents the number of dimensions (or factors) chosen, and $k \leq r$. This process is known as *dimensionality reduction*, which involves truncating all three matrices to *k* dimensions.

The reduced matrices are denoted by U_k, Σ_k, and V_k where U is a $m \times k$ matrix, Σ is a $k \times k$ matrix and V is a $n \times k$ matrix. The rank-*k* approximation to matrix A, can be constructed through $A_k = U_k \Sigma_k V_k^T$. It is important when computing the SVD that *k* is smaller than the rank *r*, because it is this feature that reduces noise in data and reveals the important relations between terms and documents [6, 3].

One common task in information retrieval systems involves a user placing a query in order to retrieve documents of interest. Given a query vector *q*, whose non-zero elements contain the weighted term frequency values of the terms, the query vector can be projected to the *k*-dimensional space using Function 1 [6].

$$Q = q^T U_k \Sigma_k^{-1}, \tag{1}$$

On the left hand side of the equation, *Q* is a mapping of *q* into latent semantic space, and on the right hand side of the equation *q* is the vector of terms in the user's weighted query; q^T is the transpose of *q*; and $q^T U_k$ is the sum of the *k*-dimensional term vectors specified in the query, multiplied by the inverse of the singular values Σ_k^{-1}. The singular values are used to separately weight each dimension of the term-document space [6].

Once the query vector is projected into the term-document space it can be compared to all other existing document vectors using a similarity measure. One very popular measure of similarity computes the *cosine* between the query vector and the document vector. Typically, using the cosine measure, the cosines of the angles between the query vector and each of the other document vectors are computed and the documents are ranked according to their similarity to the query, i.e. how close they are to the query in the term-document space. All documents or those documents with a similarity value exceeding a threshold, are returned to the user in a ranked list sorted in descending order of similarity values, i.e. the documents most similar to the query are displayed in the top of the ranked list. The quality of the results can be measured using evaluation measures, such as those discussed in Section 6. In the term-by-document matrix A that has columns a_j ($i \leq j \leq n$ where

Symbol	Name	Formula
b	Binary	$b(f_{ij})$
l	Logarithmic	$log_2(1 + f_{ij})$
n	Augmented normalised term frequency	$(b(f_{ij}) + (f_{ij}/max_k f_{kj}))/2$
t	Term frequency	f_{ij}
a	Alternate log	$b(f_{ij})(1 + log_2 f_{ij})$

Table 1: Formulas for local term-weights (l_{ij})

Symbol	Name	Formula
x	None	1
e	Entropy	$1 + (\sum_j (p_{ij} log_2(p_{ij}))/log_2 n)$
f	Inverse document frequency (IDF)	$log_2(n/\sum_j b(f_{ij}))$
g	GfIdf	$(\sum_j f_{ij})/(\sum_j b(f_{ij}))$
n	Normal	$1/\sqrt{\sum_j f_{ij}^2}$
p	Probabilistic inverse	$log_2((n - \sum_j b(f_{ij}))/\sum_j b(f_{ij}))$

Table 2: Formulas for global term-weights (g_i)

n is the number of documents in the dataset, or equivalently the number of columns in the term-by-document matrix A) the cosine similarity between the query vector $Q = (t_1, t_2, \ldots, t_m)^T$ and the *n* document vectors is given as follows:

$$cos\Theta_j = \frac{a_j^T Q}{\| a_j \|_2 \| Q \|_2} = \frac{\sum_{i=1}^m a_{ij} Q_i}{\sqrt{\sum_{i=1}^m a_{ij}^2} \sqrt{\sum_{i=1}^m Q_i^2}} \quad (2)$$

for $j = 1, \ldots, n$.

3 Background Literature

The background literature section consists of two subsections. The first subsection describes existing plagiarism detection algorithms and the second subsection describes literature on LSA applications and their parameter settings.

3.1 Source-code plagiarism detection tools

Many different plagiarism detection algorithms exist, the most popular being the *Fingerprint based algorithms* and *String-matching algorithms*. Algorithms based on the fingerprint approach work by creating "fingerprints" for each document which contain statistical information such as average number of terms per line, number of unique terms, and number of keywords [31]. An example of these is MOSS (Measure of Software Similarity) [1]. MOSS uses a string-matching algorithm which divides programs into contiguous substrings of length k, called k-grams. Each k-gram is hashed, and MOSS selects a subset of these hash values as the program's fingerprints. The more fingerprints two programs share, the more similar they are considered to be [39]. Most popular and recent string-matching based tools include JPlag [37], and Sherlock [17]. In these tools the first stage is called tokenization. At the tokenization stage, each source-code document is replaced by predefined and consistent tokens, for example different types of loops in the source-code may be replaced by the same token name regardless of their loop type (e.g. while loop, for loop). The tokens for each document are compared to determine similar source-code segments.

Moussiades and Vakali have developed a plagiarism detection system called PDetect which is based on the standard vector-based information retrieval technique [30]. PDetect represents each program as an indexed set of keywords and their frequencies found within each program, and then computes the pair-wise similarity between programs. Program pairs that have similarity greater than a given cutoff value are grouped into clusters. Their results also show that PDetect and JPlag are sensitive to different types of attacks and the authors suggest that JPlag and PDetect complement each other.

3.2 LSA parameter settings

The performance of LSA is not only driven by the SVD algorithm, but also from a variety of sources such as the corpus, term-weighting, and the cosine distance measures [42, 23]. When LSA is introduced to a new task, the parameters should be optimized for that specific task as these influence the performance of LSA. Parameters include term-weighting algorithms, number of dimensions retained, and text pre-processing techniques.

Dumais conducted experiments evaluating the information retrieval performance of LSA using various weighting schemes and dimensionality settings. LSA was applied to five information science collections (consisting of the full text of document titles, authors, and abstracts or short articles). Each dataset comprised of 82, 425, 1033, 1400, and 1460 documents and 374, 10337, 5831, 4486, and 5743 terms respectively. Dumais reported that performance, measured by Average Precision (as discussed

Symbol	Name	Formula
x	None	1
c	Cosine	$(\sum_i (g_i l_{ij})^2)^{-1/2}$

Table 3: Formulas for document length normalization (n_j)

in Section 6), improved when appropriate term weighting was applied. Normalization and GfIdf had mixed effects on performance, depending on the dataset, but on average they appear to decrease performance compared with the raw (i.e. no weighting) term frequency. Idf, Entropy and LogEntropy result in consistently large improvements in performance by 27%, 30%, and 40% respectively [11]. Nakov *et al.* [34] experimented with combinations of the weighting algorithms that were also considered by Dumais [11] and Jones [16], in order to evaluate their impact on LSA performance. Their results also revealed that local and global weight functions are independent of each other and that their performance (measured by Average Precision) is dependent on the corpus. In addition, their results revealed that, for some corpora of text, using the *logarithmic* local weighting instead of *raw* term weighting resulted in higher precision, and for others it resulted in consistently lower precision. Applying the global weighting functions *none*, *normal*, and *GfIdf*, resulted in lower precision regardless of the corpus text and local weighting function applied, and the global weight *entropy* outperformed all other global weighting functions. The results of experiments reported by Pincombe [36] concerning the performance of LSA when applying various weighting schemes are consistent with those of Dumais [11] and Nakov [34]. Their findings also show that use of a stop-word list and adding background documents during the construction of the LSA space significantly improves performance. Findings of Wild *et al.* [43] were quite different to those by Pincombe [36] and the rest of the literature discussed. They found that the *IDF* global weighting outperformed all other weighting functions, but gave no clear indication as to which local weighting function performed best. They also found that combining stemming with stop-word filtering resulted in reduced average correlations with the human scores. The findings of Wild *et al.* [43], who also investigated the correlations between LSA and human scores, were consistent with those of Pincombe [36] who found that filtering stopwords using a stop-word list improves results. Identifying the optimal number of dimensions to retain in order to best capture the semantic structure of the document collection still remains an unanswered question. With regards to the corpus size, it is well argued that more reliable results are gained from a larger corpus size [35, 38]. Rehder *et al.* [38] investigated the efficacy of LSA as a technique for evaluating the quality of student responses against human ratings, and found that for 106 student essays, the performance of LSA improved when documents contained between 70-200 words. The optimal dimensions selected by Kontostathis [21] for 7 large corpora containing between 1033 and 348,577 documents ranged from 75 to 500 depending on the corpus. Chen *et al.* [8] implemented an LSI search engine and for a collection of 600,000 documents they used 400 dimensions. Using a test database containing medical abstracts, Deerwester *et al.* [10] found that the performance of LSA can improve considerably after 10 or 20 dimensions, reaches a peak between 70 and 100 dimensions but then performance slowly diminishes. Jessup and Martin [15] also found that for their datasets a choice of dimensions ranged from 100 to 300, and Berry [3] suggests keeping at least 100 to 200 dimensions. Pincombe [36] found that, for a corpus of 50 documents, there was a major improvement in LSA performance when the number of dimensions was increased from 10 to 20, and that optimal LSA performance was achieved when no dimensionality reduction was applied, i.e. the classic VSM was used. Nakov [33] describes experiments concerned with the application of LSA to source-code programs written by Computer Science students using the C programming language. The datasets comprised of 50, 47, and 32 source-code documents. The results of the experiments revealed that LSA detected copied programs and returned relatively high similarity values to pairs containing non-copied programs. The author assumes that this was due to the fact that the programs share common language reserved terms and due to the limited number of solutions for the given programming problem. In addition, the author states that, after applying SVD, 20 dimensions were retained. Considering the size of their corpora, the choice of dimensions appears to be high, and it is suspected that this was the main reason that the authors report very high similarity values to non-similar documents. The author justifies the existence of the high similarity values to be due to documents sharing language reserved terms. However, the use of a suitable weighting scheme and appropriate number of dimensions can reduce the chances of this happening. McMillan et. al. [29] created an approach for automatically detecting closely related applications. Their tool, CLAN, helps users detect similar applications for a given Java application. CLAN is based on LSI, however the authors do not provide the parameter settings and state that weights and dimensionality were selected experimentally.

4 Contribution

Most popular plagiarism detection tools are based on string-matching algorithms. The comparison process of those approaches are based on the source-code documents structural information derived from the programming language syntax. Algorithms that rely on detecting similar

documents by analyzing their structural characteristics can be tricked by specific attacks mainly on the structure of the source-code and thus often fail to detect similar documents that contain significant code shuffling. In addition, string-matching systems are language-dependent based on the programming languages supported by their parsers [37].

LSA is an algorithm that adopts a more flexible approach than existing plagiarism detection tools, i.e. one that is not based on structural comparison and parsers. Furthermore, the similarity computation algorithms of LSA and recent plagiarism detection tools are different. One important feature of LSA is that it considers the relative similarity between documents, i.e two documents are considered similar by LSA if they are relatively more similar than other documents in the corpus, whereas, recent plagiarism detection tools compute the similarity between documents on a pair-wise basis. This is important when comparing a corpus of student solutions to a programming problem that has many similar solutions and a tool is needed to extract those similar document pairs that are *relatively* more similar than others. LSA is a highly parameterized statistical method, and its effectiveness is driven by the setting of its parameters which are set differently based on the task for which it is applied.

This paper discusses and evaluates the importance of parameterization for LSA-based similarity detection of source-code documents; and the applicability of LSA as a technique for source-code plagiarism detection when its parameters are appropriately tuned. The parameters involved in the experimentations include:

1. different corpus preprocessing steps (i.e. selective elimination of source-code elements),

2. corpus and document normalization schemes based on different weighting approaches, and

3. parameter tweaking inherent to the LSA processing, in particular, the choice of dimensions for the step of reducing the original post-SVD matrix.

Against multiple Java source-code corpora taken from a Computer Science course setting, an evaluation study on the various parameter loadings and configurations between the parameters is reported herein.

5 Experimental Setup

Experiments were performed using four Java datasets which comprised of source-code documents produced by students from the University of Warwick as part of their computer science programming courses. Ethical consent had been obtained for using the datasets. The details of these datasets are given in Table 4.

Total number of documents is the total number of source-code documents in a corpus, and *Total number of terms* is the total number of terms found in the source-code corpus after initial preprocessing is performed. During initial preprocessing, terms that are solely composed of numeric characters are removed, syntactical tokens (i.e. semicolons, and colons) and punctuation marks, and terms which occur in only one document are all removed; and upper case letters are translated into lower case. In addition, identifiers which consist of multiple terms separated by underscores are treated as single terms (i.e. after preprocessing "student_name" becomes one term "studentname"). The reason for merging rather than separating such identifiers is because each identifier represents one meaning in the source-code document, regardless whether it consists of one or two words (separated by underscore).

Total number of suspicious document pairs is the total number of document pairs that were categorised as suspicious. The following steps were carried out for compiling the set of suspicious document pairs:

1. The four corpora, one at a time, were initially passed into three source-code similarity detection tools – JPlag [37], Sherlock [17], and PlaGate [9]. The performance of JPlag is considered to be close to that of MOSS [1], however, only JPlag was available. Sherlock and PlaGate were also used because these were also readily accessible and available for performing the experiments. The output of the tools was collated and four preliminary lists (one corresponding to each corpus) were created, each containing groups of suspicious source-code documents.

2. The documents in each group were scrutinized by academics(teaching programming subjects, and who provided the particular corpora) and any false positives (i.e. documents that did not contain similar source-code to the rest of the documents in the group) were removed.

3. The final lists contained a number of queries (one random document selected from each group) and their relevant documents, and these lists were used for evaluation purposes.

Applying initial preprocessing resulted in creating four preprocessing versions, one for each dataset (A, B, C, and D), and these were named the KC version (in this version comments, keywords and skeleton code were retained in the documents). Skeleton code is the source-code provided to students as a template for completing their solutions. After the initial preprocessing, the KC version of each dataset was further pre-processed using various parameters concerning comments found in source-code, skeleton-code, and Java reserved words. The final outcome was the creation of 24 versions (six versions corresponding to each dataset: A, B, C, and D). Below is a list of abbreviations and descriptions of the preprocessing parameters used for creating each version.

– KC: Keep Comments, Keep Keywords and Keep Skeleton code

	A(RC)	A(KC)	B(RC)	B(KC)	C(RC)	C(KC)	D(RC)	D(KC)
Total number of documents	106	106	176	176	179	179	175	175
Total number of unique terms	537	1524	640	1930	348	1189	459	1408
Total number of suspicious document pairs	6	6	48	48	51	51	79	79

Table 4: The Dataset Characteristics

- KCRK: Keep Comments, Remove Keywords
- KCRKRS: Keep Comments, Remove Keywords and Remove Skeleton code
- RC: Remove Comments, Keep Keywords and Keep Skeleton code
- RCRK: Remove Comments and Remove Keywords
- RCRKRS: Remove Comments, Remove Keywords and Remove Skeleton code.

Preprocessing by *removing comments* involves deleting all comments from source-code documents such that they solely consist of source code. *Keeping comments* involves retaining source-code comments and the source-code within the documents. Experimenting with stemming or stop-word removal on the comments within the source-code documents was not conducted because the focus was mainly on preprocessing parameters within the source-code (rather than on the comments which are part of natural-language text). A list of all known *Java reserved terms* was used as a stop-list. The words contained in the stop-list were removed from all Java documents to create the relevant versions (i.e. KCRK, and RCRK). All terms found in the *skeleton documents* relevant to each corpus were added to the stop list of Java reserved terms, thus creating four new different stop lists (i.e. one for each corpus), and each stop list was applied to the relevant corpus to create the new versions (KCRKRS, and RCRKRS).

After creating the preprocessing versions, the TMG tool [45] was applied and a term-by-document matrix was created for each version. A three-letter string was used in order to represent each term-weighting scheme (as shown in Tables 1, 2, and 3) with the particular local, global and normalisation factor combinations. For example, the *tec* weighting scheme uses the *term frequency* (t) local term weighting, the *entropy* (e) global term weighting, and the *cosine document normalisation factor* (c). The following twelve local, global, and document length normalisation weighting schemes were tested: txx, txc, tfx, tfc, tgx, tgc, tnx, tnc, tex, tec, lex, lec. The literature review suggests that those weighting schemes are the most commonly used and tested by LSA researchers.

After computing the SVD of each term-by-document matrix, dimensionality reduction was performed. A range of seventeen different dimensions were tested, with k ranging from 2 to n (i.e. 2, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90, 100, n) where n is the total number of documents in a corpus. The particular dimensional settings were selected for experimentation to investigate the impact of selecting too few and too many dimensions (including the maximum number of dimensions). The datasets contained no more than 200 documents, and thus it was decided to show the effect of choosing too few and too many dimensions by evaluating the LSA performance while using a range of 2 to 100 dimensions, and also when using the maximum possible number, n, of dimensions. When maximum possible number of dimensions are used, the performance of LSA is equivalent to that of the standard Vector Space Model (VSM) [21].

The cosine similarity measure to compute the distance between two vectors was used, as it is the most commonly used measure of similarity in the literature and has been shown to produce good results.

6 Performance Evaluation Measures

The performance of a system is measured by its retrieval efficiency. *Precision* represents the proportion of retrieved documents that are relevant. Precision is denoted by P where $P \in [0, 1]$, where F_r is the number of relevant documents retrieved and F_t is the total number of documents retrieved for a given query.

$$P = \frac{F_r}{F_t} \tag{3}$$

Precision is 1.00 when every relevant document returned in the ranked list is relevant to the query. *Average Precision* (AP) is the average of the precisions of the relevant documents in the retrieved list. This evaluation measure produces a single value summary of the ranking positions of the relevant documents retrieved, by averaging the precision values obtained after each new relevant document is retrieved in the ranked list of documents. The closer the AP value is to 1.00 the better the system's retrieval performance.

A commonly used measure for evaluating the performance of an algorithm using more than one query is *Mean Average Precision* (MAP), which is the mean of the AP values of all queries. During the experiments described in this paper, the AP for each query was computed by taking into consideration the precision values of all relevant documents for the given query (i.e. no threshold was applied and thus the list of retrieved documents was not reduced). Thus AP was computed up to rank position *n*, where *n* is the total number of documents in the corpus. This is because the aim was to evaluate the rank position of all the

relevant documents for each query, and by taking into consideration the position of all relevant documents a picture of overall performance is gained. The higher the value of the MAP, the better the performance of the system, i.e. the fewer non-relevant documents exist between relevant ones.

When summarizing the behavior of a retrieval algorithm, more than a single measure is required, in order to summarize its full behavior [2]. The evaluation measures proposed by Hoad and Zobel [14] were employed as additional measures for evaluating performance, because these take into consideration similarity values between the queries and the retrieved documents. These include the *Lowest Positive Match* (LPM), *Highest False Match* (HFM) and *Separation* (Sep.). Lowest Positive Match is the lowest similarity value given to retrieved document, and Highest False Match is the highest similarity value given to a non-relevant document, in the returned list of documents. Separation is the difference between the LPM and HFM. Overall performance is calculated by taking the ratio of *Sep./HFM*, i.e. by dividing the separation by the HFM. The higher the ratio value, the better the performance.

Furthermore, for the purposes of the experiments described in this paper, a new evaluation measure *Maximum Mean Average Precision (MMAP)* is defined. MMAP is the highest MAP value achieved when using a particular weighting scheme and preprocessing parameter. For example, consider the results of the experiments presented in Tables up to 9. These tables show the MMAP values for each dataset's version.

After computing the MAP value using various weighting schemes and k dimensions, the MMAP value reached by LSA when using each weighting scheme was recorded alongside the number of dimensions that were needed for the particular MMAP value to be achieved. For example, as shown in Table 6, column KC, the highest MMAP value reached by the txx weighting scheme was 0.86 at 20 dimensions. When comparing the performance of LSA using various weighting algorithms, it is important to take into consideration the number of dimensions each weighting algorithm needed to reach its MMAP value. For example, observing the results for sub-dataset KC of dataset A, shown in Table 6, the highest MMAP recorded was that by the lec algorithm, MAP=0.96 k=40, closely followed by the tnc algorithm, MAP=0.95 k=25. The difference in MAP is only 0.01 but there is considerable difference in the number of dimensions needed, i.e. lec needed 15 more dimensions.

7 Experimental Results

This section discusses the results obtained from conducting a series of experiments for determining the impact of parameters on the performance of LSA for the task of source-code similarity detection on four source-code datasets. Section 7.1 describes the results concerned with the impact of weighting schemes, dimensionality and preprocessing settings on the applicability of LSA for detecting similar source-code documents. Section 7.2 discusses the impact of choice of parameters on the similarity values LSA assigns to document pairs.

7.1 Investigation into Weighting Schemes, Dimensionality and Preprocessing settings

Tables 6-9 show the MMAP values for each dataset's versions. Results suggest that the parameters chosen are interdependent — the performance of weighting schemes depends on the combined choice of preprocessing parameters, the corpora and the choice of dimensionality. In overall, the average MMAP values of each dataset show that the tnc weighting scheme performed well on most versions, when using between 10 and 20 dimensions. With regards to which preprocessing parameter performed best, the results vary depending on the weighting algorithm applied. When using the tnc term-weighting scheme the highest MMAP values achieved for each dataset are as follows:

- Dataset A: RC (MMAP=1.00, k=15), RCRK (MMAP=1.00, k=15),
- Dataset B: RC (MMAP=0.91, k=15), RCRK (MMAP=0.91 k=15),
  RCRKRS (MMAP=0.91, k=15),
- Dataset C: KCRKRS (MMAP=0.97, k=15), and
- Dataset D: RC (MMAP=0.92, k=5).

The results show that highest MMAP values were reached when using the tnc weighting scheme and the RC preprocessing parameter on datasets A, B and D. With regards to dataset C, highest performance (MMAP=0.97 k=15) was achieved using the tnc weighting scheme on the KCRKRS version, followed the tnc weighting algorithm on the RC version (MMAP=0.88 k=20).

Figures 1, 2, 3, and 4 show the performance of datasets A, B, C, and D respectively, when using the tnc weighting algorithm and various dimensional settings. As illustrated in Figures 1 - 4, it is clear that the choice of preprocessing parameters has a major impact on LSA performance. For example, in dataset A, Figure 1 shows that applying the RCRKRS preprocessing has a negative effect on performance, which suggests that by removing comments, keywords and skeleton code altogether important meaning from documents is also removed.

An important finding is that although the choice of preprocessing influences precision values, it does not influence the ideal number of dimensions needed for each dataset – the pattern of behavior across Figures 1 - 4 is very similar — MAP performance improves significantly after 10 to 15 dimensions and then remains steady or decreases when 35 dimensions are reached, and then begins to fall slowly and gradually. Furthermore, upon reaching the maximum number of possible dimensions (and thus the performance

of LSA is equivalent to that of the VSM), performance decreases significantly which suggests that at maximum dimensionality irrelevant information is captured by the LSA model, which causes LSA not to be able to differentiate between similar and non-similar documents.

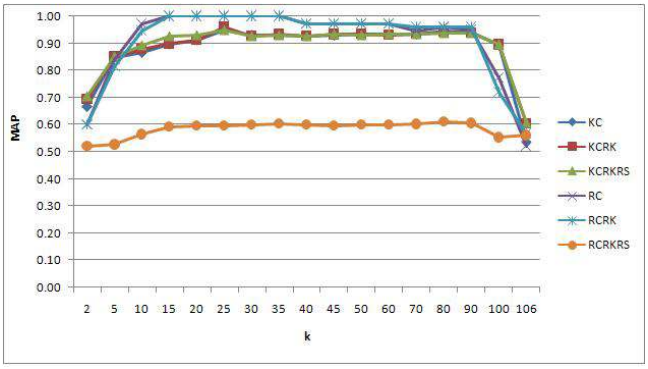

Figure 1: Dataset A: MAP performance using the tnc weighting scheme across various dimensions.

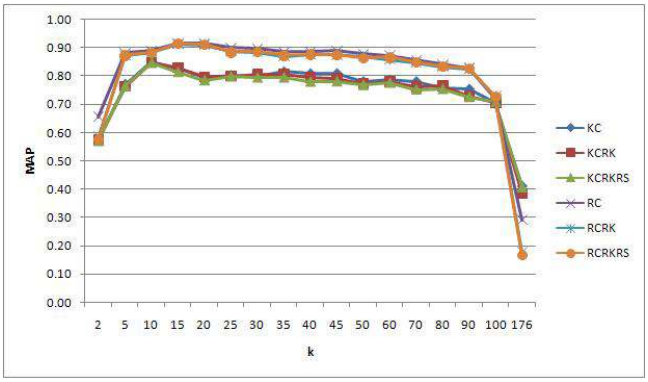

Figure 2: Dataset B: MAP performance using the tnc weighting scheme across various dimensions.

The results also show that, for the corpora involved in the experiments, when selecting between 10 and 35 dimensions, the performance of LSA outperforms that of VSM, i.e. when the value of k is set to n.

A contrast of the results was performed using MANOVA for comparing the effect of weighting schemes and preprocessing parameters on the performance of LSA. The overall performance (i.e. average MMAP and k values) was compared between the KC and all other types of preprocessing parameters, and between the txx and all other types of weighting schemes. The statistical results concerning the comparison of KC and the remaining of preprocessing parameters revealed a significant decrease in MMAP performance ($p < 0.05$, $p = 0.00$) and a significant increase in the number of dimensions ($p < 0.05$, $p = 0.04$) required for reaching MMAP when RCRKRS preprocessing is applied instead of the KC. The remaining comparisons did not reveal any statistically significant differences in MMAP performance. Thus, the statistical tests verify

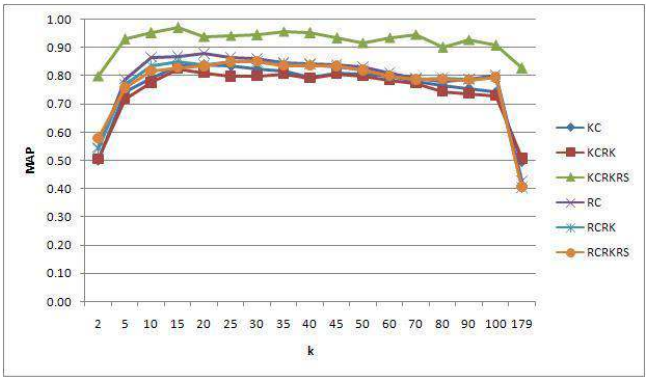

Figure 3: Dataset C: MAP performance using the tnc weighting scheme across various dimensions.

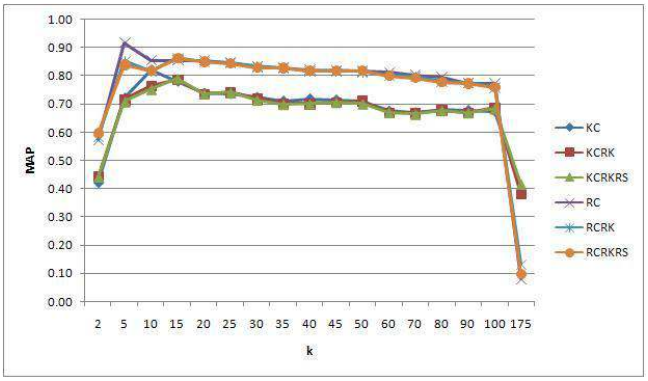

Figure 4: Dataset D: MAP performance using the tnc weighting scheme across various dimensions.

the observations that applying the RCRKRS preprocessing parameter produces undesirable effects on the retrieval performance of LSA. The most effective preprocessing parameter would achieve MMAP at less dimensions when compared to other preprocessing parameter choices - such effect had the KCRKRS and the RC settings although these results are not statistically significant.

With regards to weighting schemes, the results revealed a significant decrease in MMAP performance when the tfx ($p < 0.05$, $p = 0.03$), tgx ($p < 0.05$, $p = 0.02$), tgc ($p < 0.05$, $p = 0.02$), and tex ($p < 0.05$, $p = 0.03$) weighting schemes were applied and a significant increase in MMAP performance when the tnc ($p < 0.05$, $p = 0.02$) weighting scheme was applied. Comparisons of the performance of the txx and the remaining of the weighting schemes, did not return any statistically significant results. The statistical comparisons revealed that applying txc ($p < 0.05$, $p = 0.02$), tgc ($p < 0.05$, $p = 0.03$), lec ($p < 0.05$, $p = 0.00$) and tnc ($p < 0.05$, $p = 0.00$) significantly reduced the number of dimensions required for reaching MMAP performance. These results verify the observations gathered from Figures 1 - 4 and thus it can be concluded that the tnc weighting scheme is the most effective to apply on the LSA model for achieving maximum MMAP performance at fewer k dimensions across all datasets.

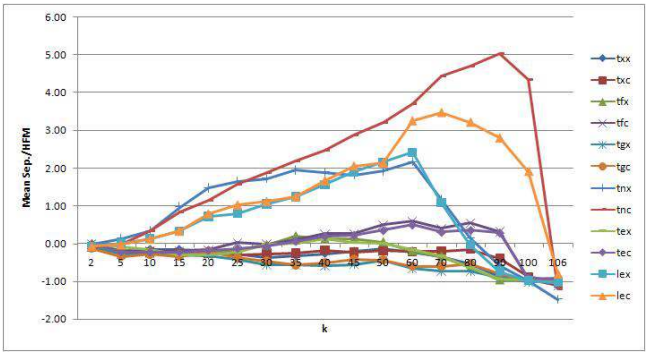

Figure 5: Dataset A version RC: Sep./HFM performance using various weighting algorithms and dimensions.

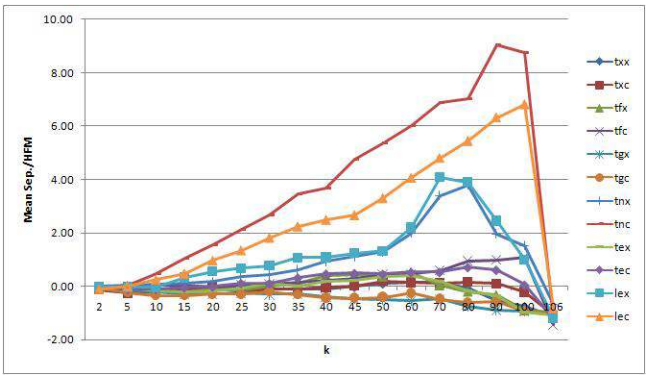

Figure 6: Dataset A version KC: Sep./HFM performance using various weighting algorithms and dimensions.

7.2 Investigation into Similarity Values

Based on the previous experiment discussed in Section 7.1, a hypothesis has been formed that parameters have an impact on the similarity values between a query and the documents in the corpus. The measure of AP does not take into consideration the similarity values between a query and the relevant documents. Importantly, when considering the most effective parameter combinations for the task of source-code similarity detection with LSA, it is also essential to investigate the similarity values assigned to similar source-code documents when various parameter settings are applied. For example, with threshold-based retrieval, if the user submits a piece of code with the aim of finding similar pieces of code that exceed the minimum similarity threshold value of 0.70, then the similarity values are an important factor in the successful retrieval of relevant documents.

However, a non-threshold based system would display the top n documents most similar to the query regardless of their similarity value with the query - for example, if the similarity value between the query and document D12 was 0.50, and 0.50 was the highest value for a relevant document for that particular query, then document D12 would be retrieved first in a ranked list of results followed by documents which received lower similarity values, with the most similar documents positioned at the top of the ranked list. Now, suppose that document D12 was indeed a relevant document and similarity threshold was set to a value above 0.60 then a threshold-based system would fail to retrieve the particular document. Thus, for the purposes of an application that detects similar source-code documents that allows use of thresholds then the similarity values are important to information retrieval performance.

The experiments performed show that similarity values are dependent on the choice of parameters. Figure 5 shows the Sep./HFM performance using various weighting algorithms and dimensions on the RC version of dataset A, and Figure 6 shows the Sep./HFM performance using various weighting algorithms and dimensions on the KC version of the same dataset. Each figure illustrates that performance is highly dependent on the choice of weighting scheme, and comparing the two figures shows that similarity values are also dependent on the choice of preprocessing parameters.

Although the best AP results were returned at 15 dimensions, evidence shows that with regards to similarity values given to relevant and non-relevant documents, 15 dimensions are too few – however, for an information retrieval task where results are ordered in a ranked list and where the similarity value does not really matter, then dimensions are appropriate for the system to retrieve the most relevant documents from the top ranked ones.

Figures 7-10 show the mean values of the LPM, HFM, and Sep./HFM, respectively, over all queries for each dataset's RC version. The average Sep./HFM values for all datasets are also displayed in each figure.

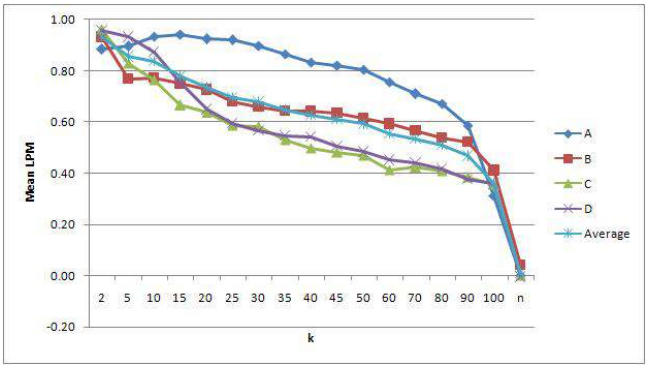

Figure 7: Datasets A, B, C, D: Mean LPM using the RC version and the tnc weighting scheme across various dimensions.

Figure 7 shows that, on average, values given to the relevant documents lowest in the retrieved list are near and above 0.80 when using 2 to 15 dimensions. In order to decide whether 15 dimensions are sufficient, the similarity values given to non-relevant documents (i.e. the HFM values) must be investigated.

Figure 8 shows that at 15 dimensions non-relevant documents received, on average, very high similarity values, i.e. above 0.70. Separation between relevant and non-relevant documents (as shown in Figure 9) is very small

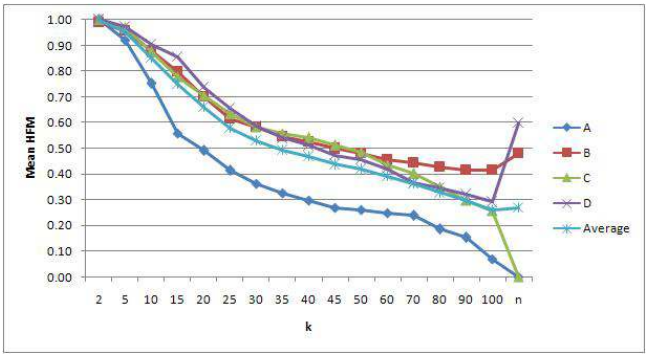

Figure 8: Datasets A, B, C, D: Mean HFM using the RC version and the tnc weighting scheme across various dimensions.

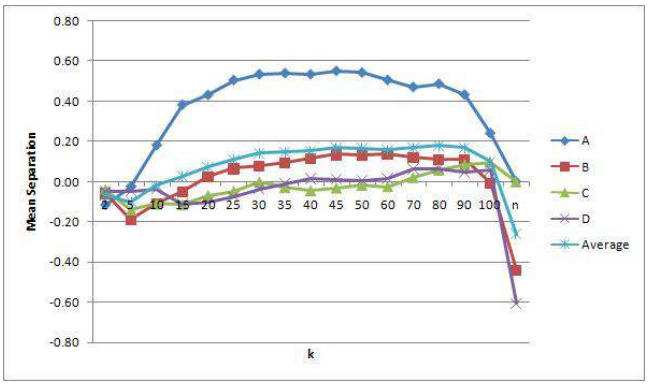

Figure 9: Datasets A, B, C, D: Mean Separation using the RC version and the tnc weighting scheme across various dimensions.

(0.03 and below) which indicates that many non-relevant documents received high similarity values.

Figure 10 shows that between 2 and 15 dimensions overall performance measured by Sep./HFM was very low (0.22 and below). These results clearly suggest that with regards to similarity values, more dimensions are needed if the functionality of filtering documents above a given threshold will be included in system implementation. At 30 and above dimensions, the average values given to non-relevant documents are 0.53 or below (see Figure 8), and there appears to be a good amount of separation (see Figure 9) between the similarity values given to relevant and non-relevant documents (i.e. average separation at 30 dimensions is 0.14, highest average separation recorded is 0.18).

With regards to good choice of dimensionality, observing the values of separation shows that there is not much change in the curve after 30 dimensions. Also, performance by means of Sep./HFM increases considerably (i.e. by 0.57 points) between 15 and 30 dimensions.

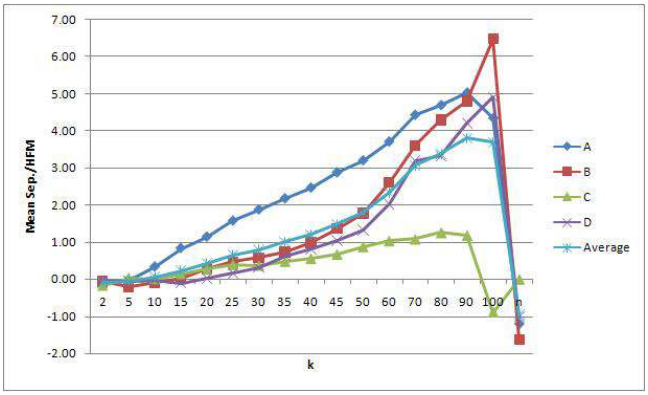

Figure 10: Datasets A, B, C, D: Mean Sep./HFM using the RC version and the tnc weighting scheme across various dimensions.

8 Discussion

The results revealed that the choice of parameters influences the effectiveness of source-code similarity detection with LSA. Most evaluations of performance in the literature are based on precision, however for LSA applications that make use of thresholds it is important to investigate the similarity values assigned to document pairs (or query-document pairs) and tune the parameters accordingly as these are crucial to the system's retrieval performance. With regards to the most effective choice of preprocessing and term-weighting parameters, the experiments revealed that removing comments during preprocessing source-code documents and applying the tnc weighting scheme to the term-by-document matrix are good choice of parameter choices for the particular application of LSA. However, the results also suggest that removing comments, Java reserved terms and skeleton code all at once can have a negative impact on retrieval performance.

In summary, the findings from the experiments revealed that applying the *term frequency* local weighting, and *normal* global weighting algorithms outperformed all other weighting schemes (with or without combining it with the *cosine document length normalization*). These results are not consistent with those by Dumais [11] who found that the *normal* global weighting performed significantly lower than all other weighting schemes (no experiments were conducted with document length normalization algorithms).

Researchers have tested various weighting schemes and best results were reported when applying the *logarithm* as the local, and the *entropy* as the global weighting scheme [34, 11, 36]. However, Wild *et al.* [43] found that the Inverse Document Frequency (IDF) global weighting outperformed all other weighting functions, and found no clear indication as to which local weighting function performed best.

With regards to source-code similarity detection with LSA, the findings described in this paper revealed that the logarithm-entropy combination performed well but only

when combined with document length normalization. On average, the optimal number of dimensions depends on the particular corpora and task (i.e. depending of whether or not threshold values will be used by the system), and for the corpora involved in the experiments the optimal number of dimensions appears to be between 10 and 30, after 30 dimensions performance begins to deteriorate.

An investigation into similarity values of document pairs revealed that choice of dimensions influences these values. The results revealed that setting the value of k to 15 dimensions is appropriate for all the source-code datasets involved in the experiments, if the task is to retrieve a ranked list of documents sorted in order of similarity to a query. However, when threshold values are used, the results suggest that the number of dimensions must be increased to 30 in order to maximize the retrieval performance of the system, and this is because when fewer dimensions were used, relatively high values were given to non-similar documents which increased the number of false positives documents being retrieved.

9 Conclusion and Future Work

This paper describes the results gathered from conducting experiments in order to investigate the impact of parameters on the effectiveness of source-code similarity detection with LSA. Results show that the effectiveness of LSA for source-code plagiarism detection is heavily dependent on the choice of parameters, and that the parameters chosen are dependent on each other, on the corpus, and on the task to which LSA has been applied. Furthermore, the results indicate that choice of dimensionality has a major impact on the similarity values LSA gives to retrieved document pairs, and that LSA based information retrieval systems which make use of threshold values as indicators of the degree of similarity between the query and documents in a corpus are likely to require more dimensions. In addition, there is clear evidence that when parameters are tuned, LSA outperforms the standard vector space model. This improvement in performance is mainly due to the use of the Singular Value Decomposition algorithm, which appears to be the power behind LSA – in fact, the vector space model is a special case of LSA, without any dimensionality reduction.

One limitation to this study was concerned with the datasets used for experimentation. The only source-code datasets that were available for conducting the experiments were those provided by academics in the department of Computer Science at the University of Warwick. It was also very time demanding to devise an exhaustive list of similar document pairs for each dataset.

Furthermore, a pattern of similar behavior was observed when using particular weighting schemes and dimensionality settings. However, this raises the question, will particular pattern of behavior change when using other source-code datasets with different characteristics (e.g. different number of documents and dictionary size)? To answer this question, one would need to conduct experiments using more source-code corpora in order to investigate the behavior of LSA's parameter settings. It would also be interesting to investigate which parameters work best by analyzing the corpora characteristics. For example, which parameters drive the effectiveness of source-code similarity detection with LSA when using C++ corpora, or corpora written in other programming languages?

Furthermore, symbols in source-code carry meaning (e.g. $y > 4$ and $y < 4$), and by removing those symbols during preprocessing, important meaning from documents may also be removed. This raises the question of, how to treat symbols in programming languages prior to applying LSA. Possible ways of answering this question would be to add the symbols to the term dictionary used to create the term-by-document matrix. Another way of treating symbols would be to replace them with words (e.g. replace symbol - with the word minus), or even to categorize symbols and to replace each one with their category name (e.g. replace occurrences of the mathematical symbols with the word arithmetic). Experiments with how to treat symbols, would be of greater importance when applying LSA to languages such as Perl, which are heavily based on symbols.

References

[1] A. Aiken. Moss: A system for detecting software plagiarism. Software: www.cs.berkeley.edu/ aiken/moss.html, accessed: July 2008.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[3] M. Berry. Large-scale sparse singular value computations. *The International Journal of Supercomputer Applications*, 6(1):13–49, Spring 1992.

[4] M. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval (Software, Environments, Tools), Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005.

[5] M. Berry, Z. Drmac, and E. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[6] M. Berry, S. Dumais, and G. O'Brien. Using linear algebra for intelligent information retrieval. Technical Report UT-CS-94-270, University of Tennessee Knoxville, TN, USA, 1994.

[7] A. Britt, P. Wiemer-Hastings, A. Larson, and C. Perfetti. Using intelligent feedback to improve sourcing and integration in students' essays. *International Journal of Artificial Intelligence in Education*, 14:359–374, 2004.

[8] C. Chen, N. Stoffel, M. Post, C. Basu, D. Bassu, and C. Behrens. Telcordia LSI engine: Implementation

and scalability issues. In *RIDE '01: Proceedings of the 11th International Workshop on research Issues in Data Engineering*, pages 51–58, Washington, DC, USA, 2001. IEEE Computer Society.

[9] G. Cosma and M. Joy. An approach to source-code plagiarism detection and investigation using latent semantic analysis. *IEEE Transactions On Computing*, 2009. Accepted for publication November 2009.

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

[11] S. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 23(2):229–236, 1991.

[12] P. Foltz. Using latent semantic indexing for information filtering. *SIGOIS Bulletin*, 11(2-3):40–47, 1990.

[13] R. Gravina, M. Yanagisawa, and K. Akahori. Development and evaluation of a visual assesment asistant using latent semantic analysis and cluster analysis. In *Proceedings of International Conference on Computers in Education*, pages 963–968, 2004.

[14] T. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, 54(3):203–215, 2003.

[15] E. Jessup and J. Martin. Taking a new look at the latent semantic analysis approach to information retrieval. In *In: Proceedings of the SIAM Workshop on Computational Information Retrieval*, pages 121–144. Raleigh, NC, 2001.

[16] K. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[17] M. Joy and M. Luck. Plagiarism in programming assignments. *IEEE Transactions on Education*, 42(1):129–133, 1999.

[18] T. Kakkonen, N. Myller, E. Sutinen, and J. Timonen. Automatic essay grading with probabilistic latent semantic analysis. In *Proceedings of the 2nd Workshop on Building Educational Applications Using Natural Language Processing at the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 29–36, Ann Arbor, Michigan, USA, 2005.

[19] T. Kakkonen and E. Sutinen. Automatic assessment of the content of essays based on course materials. In *Proceedings of the International Conference on Information Technology: Research and Education 2004 (ITRE 2004)*, pages 126–130, London, UK, 2004.

[20] S. Kawaguchi, P. Garg, M. Matsushita, and K. Inoue. Mudablue: An automatic categorization system for open source repositories. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pages 184–193, Washington, DC, USA, 2004. IEEE Computer Society.

[21] A. Kontostathis. Essential dimensions of latent semantic indexing (lsi). In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 73, Washington, DC, USA, 2007. IEEE Computer Society.

[22] A. Kuhn, S. Ducasse, and T. Girba. Enriching reverse engineering with semantic clustering. In *WCRE '05: Proceedings of the 12th Working Conference on Reverse Engineering*, pages 133–142, Washington, DC, USA, 2005. IEEE Computer Society.

[23] T. Landauer, D. Laham, B. Rehder, and M. Schreiner. How well can passage meaning be derived without using word order: A comparison of latent semantic analysis and humans. In *COGSCI-97*, pages 412–417, Stanford, CA, 1997. Lawrence Erlbaum.

[24] T. E. Lin M., Amor R. A Java reuse repository for eclipse using LSI. In *Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06)*. IEEE, 2006.

[25] M. Lungu, A. Kuhn, T. Gîrba, and M. Lanza. Interactive exploration of semantic clusters. In *3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005)*, pages 95–100, 2005.

[26] J. Maletic and A. Marcus. Supporting program comprehension using semantic and structural information. In *International Conference on Software Engineering*, pages 103–112, 2001.

[27] J. Maletic and N. Valluri. Automatic software clustering via latent semantic analysis. In *ASE '99: Proceedings of the 14th IEEE International Conference on Automated Software Engineering*, page 251, Washington, DC, USA, 1999. IEEE Computer Society.

[28] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An information retrieval approach to concept location in source code. In *Proceedings of the 11th IEEE Working Conference on Reverse Engineering (WCRE2004), Delft, The Netherlands*, pages 214–223, November 9-12 2001.

[29] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 364–374, Piscataway, NJ, USA, 2012. IEEE Press.

[30] L. Moussiades and A. Vakali. PDetect: A clustering approach for detecting plagiarism in source code datasets. *The Computer Journal*, 48(6):651–661, 2005.

[31] M. Mozgovoy. Desktop tools for offline plagiarism detection in computer programs. *Informatics in Education*, 5(1):97–112, 2006.

[32] M. Mozgovoy. *Enhancing Computer-Aided Plagiarism Detection*. Dissertation, Department of Computer Science, University of Joensuu, Department of Computer Science, University of Joensuu, P.O.Box 111, FIN-80101 Joensuu, Finland, November 2007.

[33] P. Nakov. Latent semantic analysis of textual data. In *CompSysTech '00: Proceedings of the Conference on Computer systems and Technologies*, pages 5031–5035, New York, NY, USA, 2000. ACM.

[34] P. Nakov, A. Popova, and P. Mateev. Weight functions impact on LSA performance. In *Proceedings of the EuroConference Recent Advances in Natural Language Processing (RANLP'01)*, pages 187–193. John Benjamins, Amsterdam/Philadelphia, 2001.

[35] C. Perfetti. The limits of co-occurrence: tools and theories in language research. *Discourse Processes*, 25:363–377, 1998.

[36] B. Pincombe. Comparison of human and LSA judgements of pairwise document similarities for a news corpus. Research Report No. AR-013-177, Defence Science and Technology Organisation - Australia, 2004.

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

[38] B. Rehder, M. Schreiner, M. Wolfe, D. Lahaml, W. Kintsch, and T. Landauer. Using latent semantic analysis to assess knowledge: Some technical considerations. *Discourse Processes*, 25:337–354, 1998.

[39] S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 76–85, New York, NY, USA, 2003. ACM.

[40] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29. ACM Press, 1996.

[41] A. Singhal, G. Salton, M. Mitra, and C. Buckley. Document length normalization. Technical report, Cornell University, Ithaca, NY, USA, 1995.

[42] P. Wiemer-Hastings. How latent is latent semantic analysis? In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 932–941. Morgan Kaufmann, July 31–August 6 1999.

[43] F. Wild, C. Stahl, G. Stermsek, and G. Neumann. Parameters driving effectiveness of automated essay scoring with LSA. In M. Danson, editor, *Proceedings of the 9th International Computer Assisted Assessment Conference (CAA)*, pages 485–494, Loughborough, UK, July 2005. Professional Development.

[44] L. Yi, L. Haiming, L. Zengxiang, and W. Pu. A simplified latent semantic indexing approach for multilinguistic information retrieval. In *In Proceedings of the 17th Pacific Asia Conference on Language, Information and Computation (PACLIC17)*, pages 69–79, Sentosa, Singapore, 2003. COLIPS Publications.

[45] D. Zeimpekis and E. Gallopoulos. Design of a MATLAB toolbox for term-document matrix generation. Technical Report HPCLAB-SCG, Computer Engineering and Informatics Department, University of Patras, Greece, February 2005.

LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
IDF	Inverse Document Frequency
SVD	Singular Value Decomposition
VSM	Vector Space Model
Local weighting schemes	
b	Binary
l	Logarithmic
n	Augmented normalized term frequency
t	Term frequency
a	Alternate log
Global weighting schemes	
x	None
e	Entropy
f	Inverse document frequency (IDF)
g	GfIdf
n	Normal
p	Probabilistic inverse
Document Length Normalization schemes	
x	None
c	Cosine
Preprocessing schemes	
KC	Keep Comments, Keep Keywords and Keep Skeleton code
KCRK	Keep Comments, Remove Keywords
KCRKRS	Keep Comments, Remove Keywords and Remove Skeleton code
RC	Remove Comments, Keep Keywords and Keep Skeleton code
RCRK	Remove Comments and Remove Keywords
RCRKRS	Remove Comments, Remove Keywords and Remove Skeleton code
Evaluation measures	
AP	Average Precision
MAP	Mean Average Precision
LPM	Lowest Positive Match
HFM	Highest False Match
Sep.	Separation
MMAP	Maximum Mean Average Precision
Weighting schemes (local weight, global weight, document length normalization)	
txx	Term frequency, none, none
txc	Term frequency, none, cosine
tfx	Term frequency, Idf , none
tfc	Term frequency, Idf , cosine
tgx	Term frequency, GfIdf , none
tgc	Term frequency, GfIdf , cosine
tnx	Term frequency,normal, none
tnc	Term frequency, normal, cosine
tex	Term frequency, entropy, none
tec	Term frequency, entropy, cosine
lec	log, entropy, cosine
lex	log, entropy, none

Table 5: List of Acronyms

	KC	KCRK	KCRKRS	RC	RCRK	RCRKRS	Average
txxk	0.86 20	0.86 60	0.86 60	0.78 15	0.75 40	0.54 106	0.77 50.17
txck	0.86 20	0.86 45	0.85 45	0.79 40	0.80 60	0.55 2	0.79 35.33
tfxk	0.94 40	0.92 40	0.92 40	0.91 35	0.87 45	0.61 70	0.86 45.00
tfck	0.93 70	0.94 80	0.93 80	0.88 60	0.88 60	0.60 60	0.86 68.33
tgxk	0.73 25	0.70 20	0.69 15	0.74 20	0.69 15	0.54 2	0.68 16.17
tgck	0.82 30	0.74 50	0.64 10	0.75 20	0.69 40	0.57 10	0.70 26.67
tnxk	0.92 40	0.92 40	0.92 40	1.00 35	1.00 25	0.63 70	0.90 41.67
tnck	0.95 25	0.96 25	0.95 25	1.00 15	1.00 15	0.61 80	0.91 30.83
texk	0.87 30	0.87 30	0.88 30	0.85 30	0.82 35	0.60 60	0.82 35.83
teck	0.94 80	0.94 80	0.94 70	0.87 70	0.87 60	0.61 80	0.86 73.33
lexk	0.94 20	0.93 30	0.93 30	0.97 20	0.97 25	0.62 70	0.90 32.50
leck	0.96 40	0.94 20	0.95 20	0.97 10	1.00 90	0.61 45	0.91 37.50

Table 6: MMAP values for dataset A

	KC	KCRK	KCRKRS	RC	RCRK	RCRKRS	Average
txxk	0.94 60	0.91 70	0.86 80	0.90 10	0.88 45	0.85 40	0.89 50.83
txck	0.95 15	0.88 20	0.86 15	0.90 10	0.87 5	0.60 25	0.84 15.00
tfxk	0.78 45	0.78 70	0.78 70	0.74 40	0.74 40	0.73 40	0.76 50.83
tfck	0.84 15	0.83 15	0.83 15	0.79 15	0.78 15	0.77 35	0.81 18.33
tgxk	0.92 35	0.82 60	0.77 70	0.91 25	0.88 15	0.81 40	0.85 40.83
tgck	0.92 15	0.78 20	0.74 10	0.95 15	0.89 20	0.80 20	0.85 16.67
tnxk	0.84 70	0.84 70	0.83 60	0.90 60	0.90 60	0.90 60	0.87 63.33
tnck	0.85 10	0.85 10	0.85 10	0.91 15	0.91 15	0.91 15	0.88 12.50
texk	0.80 45	0.80 45	0.80 45	0.74 90	0.74 90	0.74 90	0.77 67.50
teck	0.83 15	0.81 15	0.80 15	0.79 15	0.79 15	0.77 15	0.80 15.00
lexk	0.86 60	0.85 60	0.85 60	0.86 40	0.86 40	0.86 40	0.86 50.00
leck	0.88 15	0.88 15	0.87 15	0.90 10	0.89 10	0.87 10	0.88 12.50

Table 7: MMAP values for dataset B

	KC	KCRK	KCRKRS	RC	RCRK	RCRKRS	Average
txxk	0.78 15	0.74 15	0.98 35	0.81 90	0.77 80	0.77 90	0.81 54.17
txck	0.81 40	0.76 50	0.96 45	0.82 80	0.78 90	0.78 80	0.82 64.17
tfxk	0.65 80	0.65 70	0.91 70	0.71 70	0.71 70	0.70 70	0.72 71.67
tfck	0.73 80	0.71 90	0.94 25	0.75 60	0.70 50	0.69 50	0.75 59.17
tgxk	0.72 90	0.71 80	0.93 60	0.73 50	0.69 70	0.64 70	0.74 70.00
tgck	0.75 80	0.74 70	0.92 60	0.74 80	0.69 80	0.67 100	0.75 78.33
tnxk	0.83 25	0.79 25	0.95 25	0.82 20	0.80 35	0.79 35	0.83 27.50
tnck	0.84 20	0.82 15	0.97 15	0.88 20	0.85 15	0.85 25	0.87 18.33
texk	0.70 60	0.70 90	0.90 50	0.75 70	0.73 80	0.71 80	0.75 71.67
teck	0.73 80	0.72 80	0.96 10	0.71 60	0.70 50	0.69 80	0.75 60.00
lexk	0.74 20	0.74 20	0.96 25	0.74 35	0.74 60	0.73 60	0.78 36.67
leck	0.78 35	0.77 40	0.93 25	0.78 20	0.78 25	0.75 25	0.80 28.33

Table 8: MMAP values for dataset C

	KC	KCRK	KCRKRS	RC	RCRK	RCRKRS	Average
txxk	0.80 25	0.77 60	0.75 45	0.83 30	0.80 60	0.79 50	0.79 45.00
txck	0.82 20	0.77 20	0.76 20	0.84 30	0.80 10	0.79 10	0.80 18.33
tfxk	0.70 45	0.69 40	0.69 40	0.73 25	0.73 45	0.73 45	0.71 40.00
tfck	0.74 15	0.74 15	0.74 15	0.78 25	0.77 25	0.77 25	0.76 20.00
tgxk	0.79 30	0.73 25	0.73 25	0.81 35	0.74 70	0.73 25	0.76 35.00
tgck	0.73 30	0.70 30	0.70 30	0.79 10	0.74 15	0.73 15	0.73 21.67
tnxk	0.71 15	0.71 20	0.70 15	0.81 10	0.83 10	0.82 10	0.76 13.33
tnck	0.82 10	0.79 15	0.79 15	0.92 5	0.86 15	0.86 15	0.84 12.50
texk	0.70 45	0.70 45	0.70 50	0.74 50	0.73 40	0.73 40	0.72 45.00
teck	0.67 10	0.67 5	0.67 15	0.72 25	0.72 25	0.72 25	0.70 17.50
lexk	0.64 15	0.65 15	0.65 15	0.70 25	0.72 90	0.72 90	0.68 41.67
leck	0.76 15	0.76 15	0.76 15	0.78 20	0.78 20	0.78 20	0.77 17.50

Table 9: MMAP values for dataset D