TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

COMPARING TF-IDF AND LSI AS IR TECHNIQUE IN AN APPROACH FOR DETECTING SEMANTIC RE-IMPLEMENTATIONS IN SOURCE

Veronika Bauer, Tobias Völke, Sebastian Eder

TUM-I1527

COMPARING TF-IDF AND LSI AS IR TECHNIQUE IN AN APPROACH FOR DETECTING SEMANTIC RE-IMPLEMENTATIONS IN SOURCE CODE

VERONIKA BAUER, TOBIAS VÖLKE, SEBASTIAN EDER

Abstract. Semantic re-implementations of existing functionality are frequently reported in practice and cause increased efforts for development and maintenance. However, instances are hard to find with existing approaches. For practitioners, this increases maintenance risks, such as inconsistent bug fixing, and hinders quality improvement efforts. For researchers, this hinders a reliable quantification of the issue.

With this paper, we follow up on previous work proposing a pragmatic approach combining identifier-based concept location with static analysis to detect candidate re-implementations in source code. We present an in-depth evaluation of our approach on four experimental data sets and one industrial system. We compare two information retrieval (IR) techniques, report the respective precision and recall, and detail on the applicability of our approach in practice. In addition, we compare the findings of our approach to the findings of a clone detection.

1. Introduction

An abundance of valuable software assets is present in companies' code repositories, via Open Source libraries, and commercial component markets. Nevertheless, developers tend to re-implement existing functionality [21, 5], missing out on the benefits of reuse opportunities. Furthermore, this can result in the creation of Type-4 clones, also known as "Simions" [19], independent re-implementations of existing functionality that do not share a common origin in terms of code. These re-implementations have long term negative effects on software quality and lead to increased development and maintenance efforts.

Re-implementations can happen easily for various reasons, such as difficulties of finding or accessing reusable entities, a reuse averse development culture, or organizational limitations [5]. Prior work has addressed cases of semantic code duplication. Our notion of re-implementations is related as follows:

Semantic clones [15] are code fragments with isomorphic program dependence graphs, and therefore structurally similar. Their behaviour can, but does not need to, be functionally similar. **Accidential clones** [2] are code fragments of different origin that are syntactically similar due to the adherence to a specific protocol. This does, however, not imply behavioural similarity. **Type-4 clones** [28], **"wide miss" clones** [24], and **Simions** [19] refer to the same phenomenon: behaviourally similar code fragments that have no common origin. Unlike cloned code, these fragments are likely to differ greatly in their structure [19].

Discovering re-implementations is difficult in theory and practice: first, semantic equivalence checking is well studied (e.g. [9]) and a generally undecidable problem. Approaches to detect re-implementations therefore are constrained to resort to approximations. Second, previous work [11, 19] concludes that existing approaches, such as clone detection or random testing approaches [18], do not provide satisfactory results to detect re-implementations in practice.

Consequently, research so far is unable to realistically quantify the size of the problem. Practitioners, on the other hand, miss an approach providing support to avoid new and discover existing re-implementations [5].

To address these issues, we proposed a new approach to discover missed reuse opportunities in the form of (unintentional) re-implementations [6] in source code. Our approach uses structural code information obtained with static analysis. However, our notion of similarity is based on the concepts embodied in the source code's identifiers, accounting for the structural differences featured by re-implementations [19]. Furthermore, we reported preliminary results from a proof-of-concept implementation.

Nevertheless, one important question remained unclear: how do the results of our approach compare to the ones obtained by a clone detection? Even though re-implementations showed significant structural differences in an experimental setting, it is important to compare the two techniques on a real world system.

Problem statement: It is unclear which of the well-known IR approaches, TF-IDF or LSI, support better the detection of semantic re-implementations

Contributions: We compare two information retrieval (IR) techniques, report the respective precision and recall, and detail on their applicability in practice. We implemented our approach for Java systems and report our evaluation on four study objects.

Outline: The remainder of the paper is structured as follows: Section 2 provides a brief recapitulation of our approach before Section 3 states the goal and the research questions driving this paper. Sections 4 to 6 describe the study objects, study design and study execution. Section 7.2 reports the results, which then are discussed in Section 8. Section 9 details on threats to validity, before Section 10 provides an overview of related work. Section 11 summarizes and concludes the paper.

2. Approach

The following section presents our approach. Our approach proceeds as follows (see Figure 1): it takes as input a body of source code in which we look for re-implementations. This is a difference to our previous work, in which we required two bodies of code: on the one hand, a curated library of concept implementations to provide the concepts for the analysis, on the other hand, the study objects that was analyzed for re-implementations. From a conceptual perspective, the concept library was no longer needed for the current study: due to our previous work, the datasets available to us were more robust. This allows for a more general approach. Since in this paper we focus on the comparison of TFIDF and LSI using a proven framework [14] that uses one body of code as input, we adapted our approach to this prerequisite.

The identifiers of the code body are extracted and analyzed in a preprocessing phase to learn the specific concepts present in the source code. The preprocessed identifier information is then used in the matching phase to compute the likelihood of two code entities implementing equivalent functionality.

By resorting to identifiers, we overcome the problem of restricting similarity to a syntactic level. As studies have shown, identifiers are valuable sources for capturing programmers' intent [8, 12, 17]. Therefore, we assume that two code fragments that contain identifiers belonging to the same concept might provide the same functionality and could, therefore, be potential re-implementations.

Whilst the intuition behind this approach is quite simple, relying solely on identifiers risks to clutter the results with false positives: the same identifiers occur when defining a specific functionality as well as when using it. To mitigate this, we only consider identifiers present in *declarations* of methods, fields, and classes[1]. In this way we achieve more precision in locating the implementations of a concept.

Our approach abstracts functionality provided by identifiers on a per-file basis. We opt for this granularity to capture concepts spread over several methods[2]. During the preprocessing phase, we assign a set of "significant" identifiers to each source code file. We deem those identifiers as significant that best[3] capture the concepts of the respective file. Based on this information, we compute a similarity score for all files within each body of source code. The similarity score is based on the well known TFIDF and we compute it as follows: $\frac{\sum_{i \in I_{b \cap s}} v(i)}{\sum_{i \in I_b} v(i)}$, with I_b denoting the relevant identifiers of a concept file and $I_{b \cap s}$ denoting the overlapping relevant identifiers of a concept and a study object file. $v(i)$ denotes the weight assigned to the given identifier i.

3. Study Goal and Design

The *goal* of the study is to *evaluate* our approach for detecting semantic redundancies *for the purpose* of understanding the implications of two IR techniques *with respect to* the performance of the approach *from the viewpoint* of its applicability *in the context of* professional software development.

From this goal, we derive the following research questions:

RQ1: Which of the two IR techniques, TFIDF or LSI, is better suited for our approach? In our first prototype, we chose TFIDF as IR technique for its robustness and straightforward application. In this research question, we want to test whether TFIDF is equally suited as LSI, another widely used IR technique. We attempt an answer by reporting experiments for the following sub-research questions:

[1]We include the parameters present in the declarations.

[2]In an Object Oriented context, the assumption of concepts being captured by classes, which often can be represented in files, is a technical decision we make.

[3]"Best" is determined by the most characteristic identifiers contained in each file.

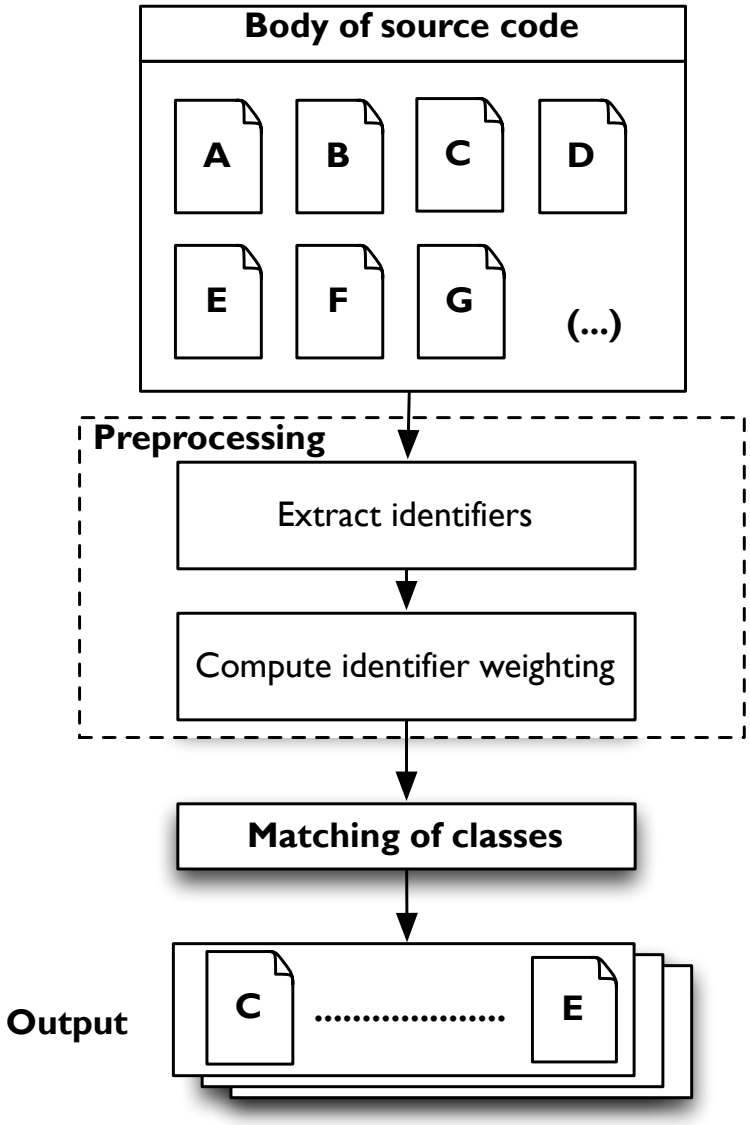


FIGURE 1. This figure illustrates our approach: we extract relevant identifiers for the concepts present in each file and compute the best matches between the bodies of code.

RQ1.1: How do precision and recall of our approach change when substituting TFIDF for LSI? In [6] we described a TFIDF-based approach supporting the discovery of potential reimplementations and tested it on a large set of software systems. Now we want to

TABLE 1. List of the study objects

Study object	Controlled, #known pairs	# LOC	# Files	Avg., Med., Min., Max. File Size	Clone Coverage	Description
AlgoDS	yes, 784	27.966	283	208, 145, 1109, 41	0.229	Collection of implementation of specific algorithms as used in [6]
ICSMEres	yes, 223	50.867	95	535, 304, 2202, 53	0.303	Results from ICSMEstart corpus in [6] as base line
DR	yes, 5565	6.904	106	65, 60, 150, 41	0.013	Results from deliberate reimplementation experiment in [19]
ICSMEstart	yes, 223	30.532.834	139.058	NA	NA	Qualitas corpus with reimplementations of collections as in [6]
Industrial	no, NA	151.571	1.194	130, 60, 2030, 5	0.255	Industrial system, excluded generated non-maintained code

provide further evaluations by determining precision and recall, as well as testing for the applicability of LSI. To answer this question we run our approach for all study objects in 4.1 with TFIDF-based as well as with LSI as IR technique. We then compare the results.

RQ1.2: How robust to noise is our approach with either of the two techniques? This questions targets the influence the amount of data not associated with other documents has on our approach, using TFIDF or LSI. Most of the use cases have a distinct set of queries (=documents), therefore there always has to be data not matching any query. We inject increasing levels of noise in our data sets and compare the quality of the results of our approach with TFIDF and with LSI. We will execute both approaches with all their variabilities on the four experimental datasets. Furthermore we will add noise to retrieve the approaches' reactions. The amount of noise documents we add is relative to the count of the datasets' documents with the percentages of 25, 50, 100 and 200%.

RQ2: How well does our approach complement the findings of a clone detection analysis? We want to explore how much additional benefit our approach gives in comparison to a clone detection analysis. We run our approach and a clone detection on one industrial system and determine the overlap of the results. Specifically, we are interested to see how many of the clone pairs are also contained in the result set of our approach and how many additional pairs of re-implementations our approach can provide.

4. STUDY OBJECTS

For our study, we run our set of analyses on four experimental datasets and one industrial system. In the following paragraphs, we detail on the characteristics of each study object. Table 1 provides a brief overview.

4.1. **Experimental Datasets.** For each of the experimental datasets, we have at disposal the source code files and a corresponding matrix containing reference links. Reference links capture pairs of source code documents that are known positives for our analysis. We store these links in a sparse matrix. For each experimental dataset, the reference matrix was validated by three researchers.

4.1.1. *Algorithms Dataset.* In the preparations of [6] we gathered implementations of of well-known algorithms of different types like e.g. graph traversing, sorting etc. These implementations provide us with very obvious links as they all implement distinctive or the same functionality. We retrieved these implementations by using an online Code Search engine [7] searching for buzzwords related to the algorithms.

4.1.2. *ICSMEres Dataset.* We manually extracted and validated results found by our original approach in [6] to gain a dataset containing source code of commonly used software systems. The Dataset consists of classes provided by the QualitasCorpus [30] (version 20130901r[4]), the Apache Commons [31], Google Guava [16] and TROVE [13].

4.1.3. *Deliberate Reimplementations Dataset (DR).* We used the source code provided by [19] to generate a very clean dataset. It consists of 109 implementations of the same functionality Juergens et al. retrieved by giving the same task to several students. As the data was validated we can assume they all implement the same functionality and therefore have to found by as positives by our analysis.

4.1.4. *ICSMEstart Dataset.* For this study we recreated the test data we used in [6]. It mainly consists out of the QualitasCorpus [30] (version 20130901r) with its various software systems. Additionally we added the Apache Commons [31], Google Guava [16] and TROVE [13]. In contrast to the ICSMEres dataset we use all of the corpus and library source code files.

4.2. **Industrial System.** Our industrial study object is a web application for resource management systems, providing service interfaces for different end user devices. It is written in Java and has been under development for the last decade, growing to 151.571 LOC distributed over ˜2900 classes. We suspected the presence of re-implementations. However, we did not have any kind of reference links or base-line at our disposal. For the analysis, we assessed the manually developed and maintained parts of the codebase, i.e. we excluded generated and non-maintained parts.

5. Study design

The following paragraphs describe the different steps we executed to conduct our study.

5.1. **Comparing the IR techniques.** To determine precision and recall in our study we chose the *Mean Average Precision (MAP)* proposed by Lohar et al. in [22]. The approach determines precision and recall based on the knowledge of reference links between different documents: It uses the set of results ordered by their similarity value and the given links, summing up the precision values of the relevant documents (given by the reference links) and divides the sum by the number of relevant documents. This provides a more accurate notion of result quality in a context where different sets of documents might constitute equally "good" results. MAP produces a value between 0 and 1.0, describing both precision and recall.

[4] `http://qualitascorpus.com/`

5.2. **Adapted TFIDF.** In this section we give a short insight into our adapted version of TFIDF and how we use it to calculate the similarities between pairs of source code.

In preparation for the comparison we compute the TFIDF value for each identifier in all the examined documents. TFIDF stands for Term Frequency Inversed Document Frequency, which is the multiplication of two values [1]. The inverse document frequency delivers us a weighting of the gained term frequencies. It makes use of the fact that a term appearing only in very few documents, or being very unequally distributed, serves as a suitable descriptor for distinguishing documents from one another [32]. According to [27] it doesn't matter which specific logarithm is used, therefore we chose the natural logarithm for our analysis.

As soon as the TFIDF values are prepared we compare every document to each other. The similarity score between each pair of documents is computed as described in Section **??**.

This analysis allows to be executed for different TFIDF thresholds. If a threshold is set, only identifiers passing this value will be considered. In [6] we discovered that this can lead to a better trade-off between finding suitable matches more precisely or simply more potential matches.

5.3. **LSI Benchmark.** LSI offers multiple parameters for tweaking. In order to evaluate its suitability for our approach, we need to determine the "best" configuration. To this end, we follow the approach described in previous work [14]: Eder et al. developed an analysis returning (amongst others) the Mean Average Precision (MAP) for a given dataset and valid links between all the documents. The Analysis provides us with different possible global and local weightings and other possible configuration options for LSI. Executing the analysis for all possible configurations allows us to find the best one for each study object.

5.4. **Data Preprocessing.** Every dataset is preprocessed as described in [6] by extracting the relevant identifiers: relevant means the identifiers of method, class and variable declarations[5]. We create two different datasets out of this data by providing one with splitted and one with unsplitted identifiers. Our splitting algorithm does not only provide us with the various tokens, but as well with the subsequences of every identifier. For example an identifier such as 'arrayInitializer_test' would result in 'array', 'arrayInitializer', 'test', 'initializerTest', and the original identifier. Before being fed to the respective analysis the split identifiers are processed by an English word stemmer.

6. Study execution

In the following paragraphs, we detail on the analysis setup and study execution for each research question.

RQ1: Which of the two IR techniques, TFIDF or LSI, is better suited for our approach?

RQ1.1: How do precision and recall of our approach change when substituting TFIDF for LSI?

[5]We use the program structure as filter on the identifiers to distinguish identifiers contained in, e.g., call relationships from those in re-implementations.

TFIDF: We execute our TFIDF based approach on all experimental datasets providing a matrix with reference links (4.1). This grants us a MAP value for every dataset and variability which we can use to make assumptions on precision and recall.

LSI: Our attempt to run our approach with a given LSI Implementation [14] on the ICSMEstart dataset, we encountered unexpected technical difficulties, that made the analysis on that dataset impossible: The original Java implementation could not handle the size of the term-document-matrix as it exceeded the maximal representable integer as its positions were persistently indexed by the library. After solving this issue, the size of the matrix caused a failure when attempting to load it into the memory on a usual machine, as well as on a server with 32 Intel Xeon E5-2650 CPU cores with 2.00 Ghz each and 64 Gb of RAM. To work around this issue, we implemented the LSI analysis, similar to the one in [14], in R, using the bigmemory package [20] to cope with the large amount of data using a file-backed Matrix. To perform the necessary SVD transformation we used the irlba package [3] which is a parallelized implementation compatible to the big.matrix of the bigmemory package. The R script outputs a ranked list of matched documents. However, despite the usage of a file-backed matrix and the server infrastructure, we were not able to successfully execute LSI on this amount of data due to memory shortage. Estimating the minimal memory consumption of running our approach with LSI on the ICSMEstart dataset 200 Gb, we stopped the attempt.

Final setup Since the ICSMEstart dataset was not suitable for comparison with LSI, we dropped it from the study execution.

RQ1.2: How robust to noise is our approach with either of the two techniques? To gradually increase the difficulty for both approaches, we furthermore add different percentages of noise to the datasets. The exact number of documents added is relative to the original documents in the dataset. For this study we added noise at the rates of 0, 25, 50, 100 and 200%. For the noise dataset we used core components as well as the clone detection module from the ConQAT system[6] as it provides us with highly specific functionality that is unlikely to contain re-implementations. Additionally, we manually validated and cleansed the source code from classes similar to the ones in our test data. Additionally to the noise we add a noise catcher file which contains one single unsplittable identifier not contained by any other document. Our approach processes the noise source code equally to rest of the documents. During the analysis, we dynamically add links between the noise files and a noise-catcher file to allow the MAP algorithm to identify them as false positives regarding to the rest of the documents. Executing our approach with both techniques for all variabilities and noise percentages provides various MAP values allowing us to evaluate their vulnerability to noise.

RQ2: How well does our approach complement the findings of a clone detection analysis? For all study objects, we run the ConQAT clone detection (for Type 1 to 3) with a minimum clone length of 10. We report the clone coverage for each study object and compare the reported clone pairs with the re-implementation pairs found by our approach using TFIDF and LSI, respectively. Furthermore, we manually inspect the

[6] `www.conqat.org`

results for the industrial system to assess the result quality of the respective techniques. Additionally, two system experts assess the positive hits of our approach for their practical relevance.

7. Results

In this section we report on result analysis and the results for our RQs.

RQ1: Which of the two IR techniques, TFIDF or LSI, is better suited for our approach? For the experimental data sets, we had at hand a baseline of re-implementations that all of the authors reviewed and validated. This served as basis for the result analysis. *RQ1.1: How do precision and recall of our approach change when substituting TFIDF for LSI?*

For this RQ, to provide a valid comparison between the approaches, we used the results of the best 20% of the configurations for LSI and TFIDF.

Table 2 shows the best MAP values we retrieved with TFIDF and LSI for the experimental datasets. TFIDF produces as best results a MAP of 0.8203 for the algorithms dataset and 0.9316 for the ICSMEres dataset. The results for the algorithms dataset are wider spread than the ones from the ICSMEres dataset.LSI produces a MAP of 0.8989 for the algorithms and 0.8675 for the ICSMEresults datasets. For the DR dataset set, TFIDF and LSI delivered a MAP value of 1.0 for all configurations.

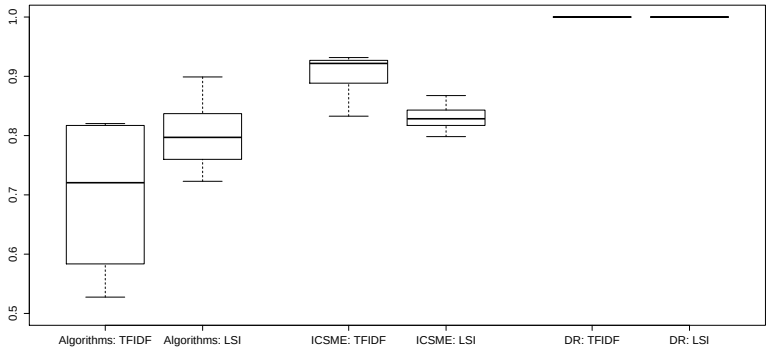


Figure 2. Boxplot of the top 20 % MAP values retrieved by using TFIDF and LSI as IR approach.

Figure 2 shows boxplots for the best 20% of the results of using the different IR techniques on the experimental datasets (without noise). Whilst the best values for MAP in the datasets are very similar for both techniques, the values of the best 20% of all configurations show a wider spread for TFIDF.

RQ1.2: How robust to noise is our approach with either of the two techniques? To answer this question we executed both approaches injecting different amounts of noise in terms of functionality unrelated to the respective datasets. Figure 3 depicts these results

TABLE 2. Best MAP Value for LSI and TFIDF per experimental dataset

			LSI		**TFIDF**	
Dataset	w_l	w_g	Cut-Off Rank	MAP	MAP	Thresh.
Algorithms	ltf	H	12	0.8989	0.8203	1
ICSMEres	ltf	H	18	0.8675	0.9316	2
DR	all	all	all	1.0	1.0	all

grouped by the dataset and the technique. The values underlying these plots consist of the best 20% of every configuration. Examining the boxplots for the algorithms dataset one can see, that LSI delivers a slightly decreasing but nearly stable result. In contrast, TFIDF already starts with a wider spread and deteriorates faster. Taking into consideration the best values for every approach as stated in Table 3, LSI furthermore shows a better result for every noise percentage.

The ICSMEresult dataset led to opposite results: While the LSI results deteriorate with increased noise, TFIDF seems to be hardly affected. The peak value (without noise) for the LSI approach is close to the ones of the algorithms dataset. TFIDF, however, produces a higher MAP than LSI on this dataset.

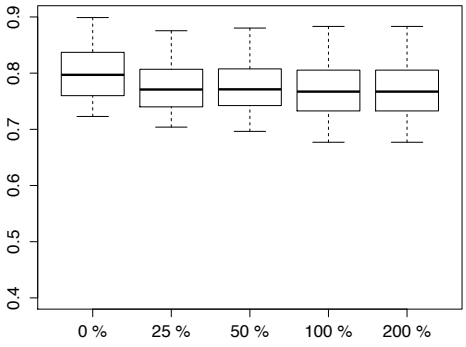
With respect to the DR Dataset we get a completely different picture. While the best values, as depicted in Table 3, are all perfect with a MAP value of 1.0 for both techniques, the TFIDF results drop drastically with increased noise whilst the LSI MAP is only slightly decreasing.

TABLE 3. Best MAP value for varying noise percentages per dataset and IR technique

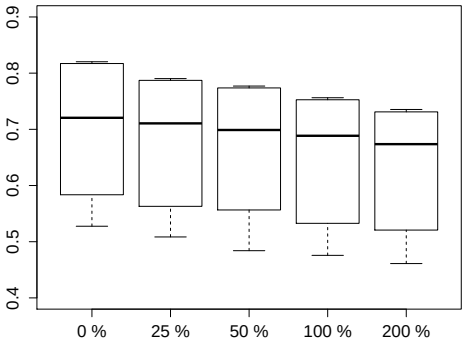
Datasets	**Algorithms**		**ICSME**		**DR**	
Noise %	LSI	TFIDF	LSI	TFIDF	LSI	TFIDF
0	0.8989	0.8203	0.8675	0.9316	1.0	1.0
25	0.8755	0.7903	0.8554	0.9307	1.0	1.0
50	0.8802	0.7770	0.8518	0.9303	1.0	1.0
100	0.8832	0.7562	0.8323	0.9292	1.0	1.0
200	0.7739	0.7353	0.7994	0.9296	1.0	1.0

RQ2: How well does our approach complement the findings of a clone detection analysis? We answer the RQ in two steps: first, we describe our findings on the experimental datasets by relating our reference links to the results of the clone detection. Second, we analyze the results of the two IR techniques and clone detection on the industrial study object.

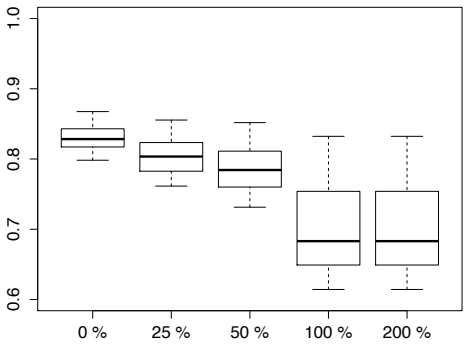
7.1. **Experimental datasets.** For the experimental datasets, we captured how many of the reference links of each dataset were also found by the clone detection. For the Algorithms dataset, clone detection discovered 27 out of 784 known pairs, for the DR dataset, 1 out of 5565 known pairs, and for the ICSMEres dataset, 16 out of 223 known pairs.



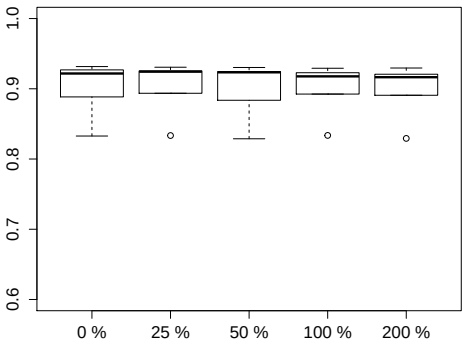
(a) Algorithms Dataset: LSI.



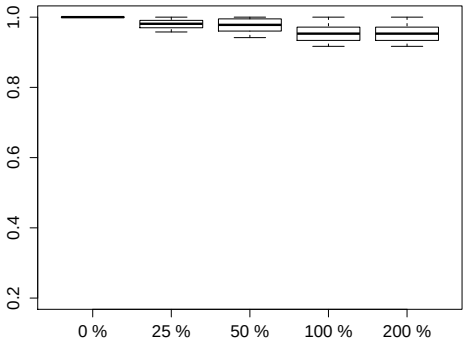
(b) Algorithms Dataset: TFIDF.



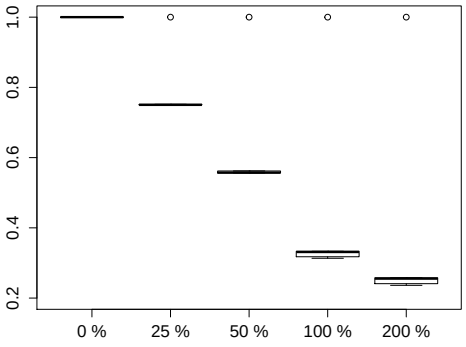
(c) ICSMEres Dataset: LSI.



(d) ICSMEres Dataset: TFIDF.



(e) DR Dataset: LSI.



(f) DR Dataset: TFIDF.

FIGURE 3. Results of the best 20% of the MAP values of our approach using LSI (left) and TFIDF (right) on the Algorithms, ICSMEres, and DR datasets with noise levels of 0, 25, 50, 100, and 200%.

7.2. **Industrial study object.** For this RQ, we analyzed our industrial study object with both versions of our approach using TFIDF and LSI, respectively, as well as with a clone detection. For the industrial system, we did not have a baseline of reference links at disposal. Therefore, we had to analyze the results manually. However, each of our setups produces a significant number of results (>15000 hits). For feasibility reasons, we had to limit the number of result pairs. We proceeded as follows: for each of the IR techniques as well as for clone detection, we sorted the results of the analysis in descending order[7]. Then, we manually assessed the top 200 result pairs for each sorting. Furthermore, with a system expert, we judged the relevance of the result pair in terms of it requiring to be acted upon to improve the system quality.We manually analyzed the top 200 hits produced for each setup to determine if it was a re-implementation. Then we intersected the top 200 hits per pairs of techniques and of all techniques. Figure 4 shows a proportional Venn diagram of these intersections and Table 4 summarizes the results. Furthermore, we distinguished between relevant and irrelevant re-implementations: relevant re-implementations indicated the potential for action to improve the system. Relevant hits were re-implementations that did not originate from deprecated redundancies or were not imposed by system conventions. In contrast, redundancies caused by system conventions, such as exception handling, or by implementations of interfaces or deprecated functionality were not considered as relevant hit.

When applying each technique by itself on the system, more than half of the resulting pairs were irrelevant hits for LSI and TFIDF. Clone detection in this case performed better than either approach. When intersecting the results of the different techniques, TFIDF and LSI had the largest overlap (91 pairs) of which 78% were positive hits. The overlaps between CD and TFIDF (35 pairs, 91% positive hits) and CD and LSI (33 pairs, 100% positive hits) produced far smaller result sets. However, the quality of the results were greatly improved. Intersecting the top 200 results of all three techniques produced 28 result pairs, which were all positive hits.

8. Discussion

In this section we answer our RQs based on the results in the former section.

RQ1.1: How do precision and recall of our approach change when substituting TFIDF for LSI? Examining Table 2 we can see that our approach performs well for both techniques, delivering MAP values from 0.8 to 1.0 (given the best configuration for each approach). As to the 'perfect' result of the DR dataset is of limited significance to RQ1.1. The nature of the MAP calculation only allows us to evaluate the quality of the approach by the order of its results. Therefore, only containing positives in the dataset, we cannot get any value worse than 1.0. However, this changes with noise (see RQ1.2).

The ICSMEres dataset's results are the best of the two remaining. This was to be expected, as we used the same approach to collect the contained reference links.

Regarding the problem of deliberate implementations of similar algorithms (represented by the Algorithms dataset), we can see that, at least for the best configurations, our

[7]For the clone analysis, we summed up the number of cloned units per file.

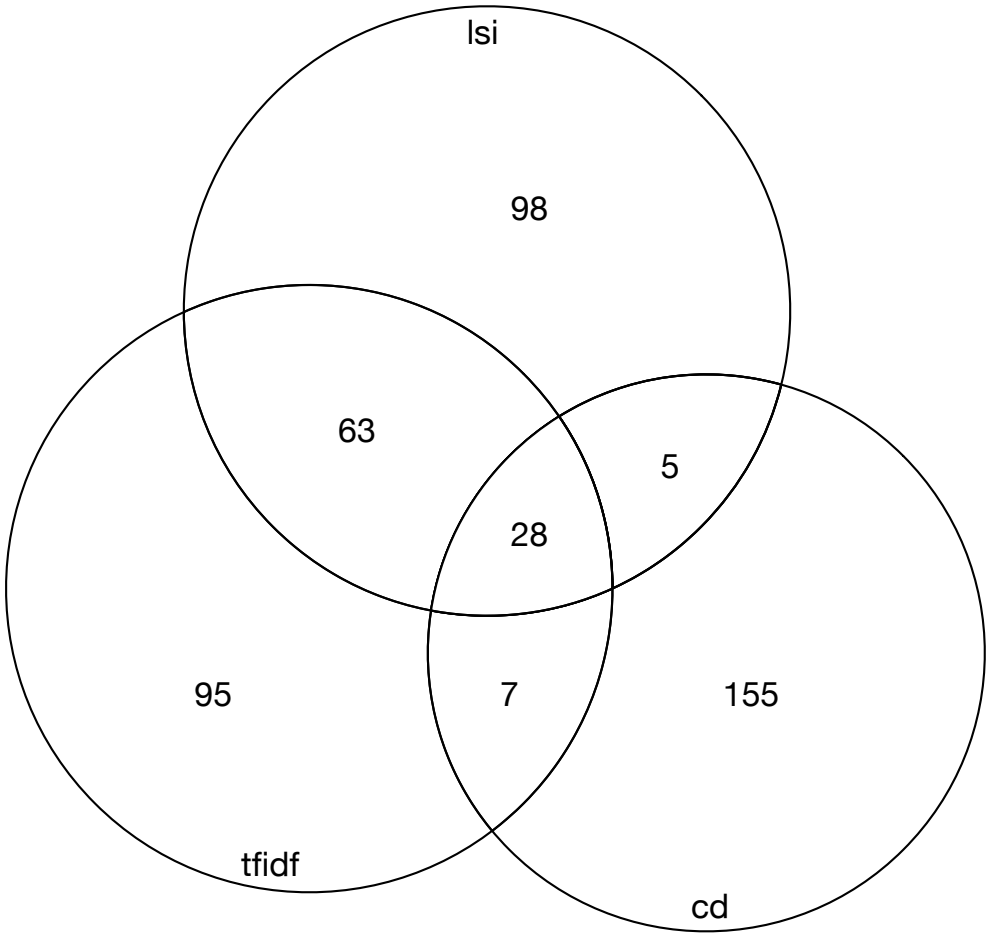


FIGURE 4. Scaled Venn diagram approximating the overlaps of the results of the top 200 results or our approach with TFIDF and LSI vs. the ConQAT clone detection.

approach appears to be suitable for this type of problem. This is further supported by the ICSMEres dataset's results.

Examining Figure 2 and Table 2 both approaches give a similar performance. Although on different datasets one is better than the other, all in all the provided results are satisfying. LSI has an average MAP (considering only the best results for each dataset without noise) of 0.9221 and TFIDF of 0.9173.

RQ1.2: How robust to noise is our approach with either of the two techniques? For this RQ we examined the effect of added noise on the results of our approach using TFIDF or LSI, respectively. The boxplots in Figure 3 provide an overview of the resulting MAP

TABLE 4. Qualitative assessment of the results of the different techniques on the industrial system.

	Industrial		
Assessment of results	LSI	TFIDF	CD
# of positives	98/200	87/200	150/200
# of relevance	64/200	58/200	114/200
	LSI intersected with TFIDF		CD
# of overlaps	91		
# of positives	71/91		
# of relevance	50/91		
	LSI	TFIDF intersected with CD	
# of overlaps		35	
# of positives		32/35	
# of relevance		24/35	
	LSI intersected with CD		
# of overlaps	33		33
# of positives	33/33		33/33
# of relevance	23/33		23/33
	Intersection of the three techniques		
# of overlaps	28		
# of positives	28/28		
# of relevance	19/28		

value ranges for different amounts of noise. On the Algorithms dataset, LSI performs very well as increasing the noise levels shows only little effect on the results. TFIDF as well provides good results, but worsens as more noise is added.

On the ICSMEres dataset, the opposite happens. Although both approaches return satisfying results, LSI reacts stronger to the noise. In this case TFIDF stays nearly constant. However, one has to keep in mind that the that the reference links of this dataset were established by means of the original version of our approach based on TFIDF. This means that LSI could be penalized in terms of the result. Under this aspect, LSI still performs reasonably well.

For the Algorithms and ICSMEres dataset, both approaches deliver a good precision and recall for their best configurations as shown in Table 2. While the variance of the results may be increasing, the noise has little effect on the result for the most suitable configuration.

With respect to the the DR dataset the best results (Table 2) for all noise percentages are all 1.0. Nevertheless, LSI performs with more stability its different configurations. Our TFIDF only performs as expected for one configuration, strongly worsening otherwise with increasing noise.

Both approaches perform well, even when hardening the problem with noise. For the DR dataset, our TFIDF approach appears less reliable.

RQ1: Which of the two IR techniques, TFIDF or LSI, is better suited for our approach? For the comparison of the LSI and the TFIDF approach in this section we rely on the answers from RQ1.1 and RQ1.2. Both techniques showed good performance with and without added noise. Compared to each other, LSI is performing better, taking into consideration that the ICSMEres dataset is easier to analyze for TFIDF. This conclusion is supported by the wide spreading of the MAP values provided by applying TFIDF on the DR dataset. However, when attempting to run our approach on larger datasets, such as ICSMEstart, LSI runs into scalability issues, whereas our original TFIDF-based approach scales up.

Taking into consideration the reaction to noise as well as the fact that the ICSMEres dataset is easier to analyze for TFIDF we assume that, in general our approach performs better when using LSI as IR technique.

RQ2: How well does our approach complement the findings of a clone detection analysis? For the experimental datasets, the overlap between the given reference links and the results of the clone detection is small (between 0 and 7%), which indicates that both analyses could complement each other.

For the industrial system, LSI as well as TFIDF in our approach produced a large number of results. However, during the manual analysis, we found that findings due to system conventions, interface implementations, and deprecated units cluttered even the top 200 results and decreased the number of relevant re-implementations.

We were interested to see whether LSI, TFIDF, and clone detection ranked the same results in the first 200 positions. As the intersection between the results of LSI and TFIDF shows, this applies for less than 50% of the results. However, the majority of positives and the majority of relevant hits are preserved. Intersecting the results of the two IR techniques thus increases the result quality and relevance.

The intersection of the results of a clone detection on the industrial system with either of the IR techniques resulted in significantly smaller sets of overlapping pairs (35 for TFIDF-CD, 33 for LSI-CD). Therefore, we conclude that our approach and clone detection deliver different kind of results and are, therefore, complementary. The quality of the remaining results is between 91 and 100%.

Intersecting the top 200 results for all three analyses only marginally reduces the number of result pairs to 28. Again, all reported pairs are positive hits.

As far as the types of results are concerned, we found during manual inspection that intersecting our approach with clone detection had the effect of propagating smaller cloned units with semantic overlap to a significantly better position in the result set. Since these were considered as relevant by practitioners, the intersection had the effect of providing them with a focussed result set that they were willing to assess.

Summarizing, we can conclude that our approach is complementary to clone detection. In addition, we find that intersecting the top results of our approach with LSI as the IR technique with the top results of clone detection, we can produce a more focussed set of high quality results than either of the analyses by itself.

9. Threats to Validity

Human Error. All experimental datasets are possibly flawed as the assumptions made on the true positives rely on our subjective judgment. It is likely that we did not find every single reference link between documents containing re-implementations. Furthermore, some of the links we considered as positives may be too general or not similar due to misunderstandings about the functionality. To mitigate these risks, we applied researcher triangulation and assessed the reference links independently.

Selection of Study Objects. The ICSMEres dataset is slightly biased towards using TFIDF in our approach, as the reference links between the contained documents were found by means of the original version of our approach presented in [6]. Therefore, we expect our original approach to have an advantage over the LSI-based one.

Our experimental datasets, furthermore, are curated and thus do not necessarily reflect the characteristics of a real world software system. Especially the DR dataset containing 109 re-implementations of the same functionality obviously is far from any problem in the real world. We attempted to mitigate this threat with two measures: First, we incrementally added noise to the different datasets and, in this way, enhanced them with source code which appeared in other systems. Second, we applied our approach on a real world system.

10. Related Work

In the following, we present approaches that aim to detect or avoid untintentional re-implementations.

10.1. **Detecting similar implementations.** Closest to our approach is the work by Marcus and Maletic [24]: they aim to interactively detect *high-level concept clones* by computing the similarity of source code documents (that can be of the granularity of files or methods) and clustering of the results. The similarity is computed with Latent Semantic Indexing, LSI [10]. The clustering can be enhanced by using structural information. Determining relevant high-level concepts is done by the user. In a case study, the authors uncover redundant implementations of a list within one system. Our contribution differs in scope and techniques: We provide a comparison of two IR techniques, LSI and TFIDF, for our approach, and relate the findings of our approach to the results of a clone detection.

Al-Ekram et al. [2] report empirical findings on *accidential cloning* across software systems. Their approach detects structurally similar code fragments caused by usage patterns required by specific technologies. However, the authors state that their approach is likely to miss re-implementations that are fundamentally different in structure.

Jiang and Su [18] propose random testing to automatically mine functionally equivalent code fragments. The source code is randomly cut in chunks. Two chunks are considered equivalent if they produce the same output for the same random input data. The study reports promising results for the test systems, namely a Linux Kernel and a sorting

benchmark. However, Deissenboeck et al. [11] found that reproducing Jiang and Su's experiment on Java code yielded unsatisfactory results that provided little value for practical application.

Kawrykow and Robillard [21] propose an approach to mine Java systems for methods "imitating" library methods available to these systems. Their goal is to replace functionality implemented in the client code by calls provided by the library. They abstract method bodies to program elements and perform a matching between the available library methods and the client methods. Whilst they cater to the important use case of replacing obsolete client methods by library methods, the notion of equivalence on the method level is still too restrictive for our task: the re-implementations we are looking for might be present in different code structures and would therefore be missed by the approach.

10.2. **Concept location.** The use of identifiers is a common strategy for concept location [23]. Methods from text retrieval, TR, (such as Term Frequency-Inverse Document Frequency, TF-IDF [29], or Latent Semantic Indexing, LSI [10]) are used to extract concepts given by e.g. use cases from source code. Recently, these approaches have been enhanced by adding static and/or dynamic program information. A study by Basset and Kraft [4] further suggests that structural term weighting can improve TR based concept location. To the best of our knowledge, the mentioned techniques have not been applied to our case.

Amongst the existing IR techniques, LSI is one of the most frequently used and best documented. In the following we want to state different approaches similar to ours using this and related techniques.

Natural text to Source Code. Most IR algorithms are based on having a defined query and a set of documents to search on. In [25] Marcus et al. used LSI for concept location. They let programmers write queries to map them to relevant parts in the source code. The approach described by Poshyvanyk and Marcus in [26] is a combination of Formal Concept Analysis and LSI. The queries are still created manually, but the approach organizes the results in a concept lattice. This is further supported by a recommender system providing similar words for the query.

Source Code to Source Code. In [24], Marcus and Maletic provide a segment of source code implementing a specific concept. They use the contained comments and identifiers to create a query they give to an LSI analysis then tracing the rest of the source code. Using Latent Semantic Analysis combined with signature matching Ye and Fischer created a tool providing the programmer with suggestions while implementing new Code [33]. It queries the existing implementations in real time while the user enhances the information by providing more and more source code.

11. Conclusions

Acknowledgements

Parts of this work were funded by the Federal Ministry of Education and Research, Germany (BMBF). Project code Software Campus (TU München), grant number 01IS12057.

References

[1] A. Aizawa. An information-theoretic perspective of tf-idf measures. *Information Processing & Management*, 39(1):45–65, 2003.
[2] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey. Cloning by accident: An empirical study of source code cloning across software systems. In *ISESE*, 2005.
[3] J. Baglama and L. Reichel. irlba: Fast partial SVD by implicitly-restarted Lanczos bidiagonalization. `http://cran.r-project.org/web/packages/irlba/index.html`.
[4] B. Basset and N. A. Kraft. Structural Information Based Term Weighting in Text Retrieval for Feature Location. In *ICPC'13*.
[5] V. Bauer, J. Eckhardt, B. Hauptmann, and M. Klimek. An exploratory study on reuse at google. In *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices*, pages 14–23. ACM, 2014.
[6] V. Bauer, T. Völke, and E. Jürgens. A Novel Approach to Detect Unintentional Re-implementations. In *ICSME'14*, 2014.
[7] BlackDuck. Black Duck Open Hub Code Search. `http://code.ohloh.net/`.
[8] B. Caprile and P. Tonella. Restructuring program identifier names. In *ICSM'00*, 2000.
[9] G. Cousineau and P. Enjalbert. Program equivalence and provability. In *MFCS*, 1979.
[10] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. 1990.
[11] F. Deissenboeck, L. Heinemann, B. Hummel, and S. Wagner. Challenges of the dynamic detection of functionally similar code fragments. In *CSMR'12*, 2012.
[12] F. Deissenboeck and M. Pizka. Concise and consistent naming. *Software Quality Journal*, 14(3):261–282, 2006.
[13] R. Eden. TROVE: High Performance Collections for Java. `https://bitbucket.org/robeden/trove/downloads`, 2013. accessed 04-November-2013.
[14] S. Eder, H. Femmer, B. Hauptmann, and M. Junker. Configuring latent semantic indexing for requirements tracing. *RET 2015 : 2nd International Workshop on Requirements Engineering and Testing*, 2015.
[15] M. Gabel, L. Jiang, and Z. Su. Scalable detection of semantic clones. In *ICSE' 08*.
[16] Google. Guava: Google Core Libraries for Java 1.6+. `http://code.google.com/p/guava-libraries/`, 2013. accessed 04-November-2013.
[17] S. Haiduc and A. Marcus. On the use of domain terms in source code. In *ICPC'08*.
[18] L. Jiang and Z. Su. Automatic mining of functionally equivalent code fragments via random testing. In *ISSTA*, 2009.
[19] E. Jürgens, F. Deissenboeck, and B. Hummel. Code similarities beyond copy & paste. In *CSMR*, 2010.
[20] M. J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013.
[21] D. Kawrykow and M. P. Robillard. Improving API Usage through Automatic Detection of Redundant Code. In *Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on*, pages 111–122. IEEE, 2009.
[22] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 378–388, 2013.

[23] A. Marcus and S. Haiduc. Text retrieval approaches for concept location in source code. In *ISSSE'11*.
[24] A. Marcus and J. I. Maletic. Identification of high-level concept clones in source code. In *ASE'01*, 2001.
[25] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An Information Retrieval Approach to Concept Location in Source Code. *11th Working Conference on Reverse Engineering (WCRE04)*, 11, 2004.
[26] D. Poshyvanyk and A. Marcus. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code. *IEEE 15th International Conference on Program Comprehension (ICPC'07)*, 15, 2007.
[27] S. Robertson. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004.
[28] C. K. Roy and J. R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEEN'S UNIVERSITY*, 115, 2007.
[29] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval.
[30] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. *2010 Asia Pacific Software Engineering Conference*, pages 336–345, 2010.
[31] The Apache Foundation. Apache commons.
[32] S. K. M. Wong and Y. Yao. An information-theoretic measure of term specificity. *Journal of the American Society for Information Science*, 43(1):54–61, 1992.
[33] Y. Ye and G. Fischer. Information delivery in support of learning reusable software components on demand. In *IUI*, pages 159–166, 2002.