Information processing for Similar Source code using LSH Algorithm

*Mrs Vani Dave[1],Mr Sanjeev Kumar shukla[2]

[1]M.Tech Research Scholar, Department of Computer Science and Engineering, Kanpur Institute of Technology , Kanpur, India

[2]Assistant professor and Head of Department, Department of Computer Applications, Kanpur Institute of Technology , Kanpur, India

[1]vani_dave@rediffmail.com , [2]sas@kit.ac.in

Abstract

In this study, we propose a method to quickly search for similar source files for a given source file as a method to examine the origin of reused code. By outputting not only the same contents but also similar contents, it corresponds to the source file that has been changed during reuse. In addition, locality-sensitive hashing is used to search from a large number of source files, enabling fast search .By this method, it is possible to know the origin of the reused code .A case study was conducted on a library that is being reused written in C language. Some of the changes were unique to the project, and some were no longer consistent with the source files. As a result, it was possible to detect the source files that were reused from among the 200 projects with 92% accuracy. In addition, when we measured the execution time of the search using 4 files, the search was completed within 1 second for each file.

Keywords: LSH, n-gram and Jaccard coefficients, MinHash Value, TF-IDF

1. Introduction:

In software development, software developed by other projects is frequently reused .If you need project-specific changes or want to simplify the compilation process, reuse is done by copying and importing the source code of the source project. While there are advantages to such reuse, there is a risk that defects contained in the reused code will be captured .In order to manage the reused code, information about which project and version of the reused code is needed. However, it has been found that not all projects record such information. [20]

If there is a match in another project for the reused source file, it can be said that the source file may be derived from that project. However, code changes may be made during reuse, and in that case it is not enough to simply determine whether the source files match. Inoue et al. [8] proposed a system called the Ichi Tracker. This system queries code fragments and searches using a code search engine. Also, in our group, we target the source code in the repository, the version In this paper, we propose a new method to detect the presence of an ion. [12]This approach requires that you have a repo to reuse from In this study, we propose a fast search method for similar source code for a given source file. locality-sensitive hashing [7] is used to speed up search. In the proposed method, the source files to be searched are first registered in the database. In this case, in order to use the LSH algorithm, the source file is converted to a vector consisting of MinHash[2]values. When performing a search, the target source file is converted to a vector as well, and the search is performed using the vector .

In order to evaluate the method, the validity of the method was evaluated using 2 cases. In addition, as a case study, we applied the method to the code that was actually reused, and evaluated whether the recorded source of reuse can be detected. The time required for the method was also evaluated.

In chapter 2, related research is described. Chapter 3 describes the proposed method in this research, and chapter 4 describes the implementation in this research. Chapter 5 describes the case studies conducted, and Chapter 6 describes threats to validity. Finally, Chapter 7 is a summary of this research.

2. Related research:

2.1 Reuse of Source Code: When developers develop software, they reuse software developed in other projects. Heinemann et al. [6] surveyed open-source Java projects and found that black-box is a binary reuse. It was reported that reuse was dominant over the source code reuse, which is the white-box reuse. Rubin et al. [18] also reported that companies are reusing developed software for use in the development of new products.

It is possible to develop software efficiently by reducing the cost of development by reusing it. It has been reported by Mohagheghi et al. that reused components are less defective than those that are not. [15]

If a developer needs to make changes to the project's own when reusing it, it may be possible to incorporate the source code into the project under development. For example, v8monkey*1 adds modifications to the libpng * 2 library to support the APNG (Animated Portable Network Graphics)format.

While reusing source code is useful, developers should be careful not to use those that have bugs or security vulnerabilities. Xia et al. [20] investigated whether the library being used is a potentially vulnerable version of a project that reuses code from an open source library. As a result, 31.1% of projects using zlib, 85.7% of projects using libcurl, and 92% of projects using libpng were found to use potentially vulnerable versions. In addition, 18.7% of projects did not have any information about which version of the library they used. In addition, 4.9% of projects have changed directory names or mixed source code with reused source code, making it difficult to manage. It is considered useful to identify the source of reuse for such unmanaged projects.

The detection of software reuse is based on binary or source code. For binary applications, Davies et al. [4] and [5]proposed a method for using signatures in a class from Java binaries. S. bj rnsen et al. [19]proposed a method for detecting code clones from executable binaries. Qiu et al. [17] proposed a method for identifying library functions from binaries.

Inoue et al. [8] proposed a system called Ichi Tracker. It queries code fragments, etc., and searches the source code including clones of the code fragments using a code search engine. Along with the clones found as the output of the system, for each clone the last modified date of that clone It provides information such as the code of the project, the content of the query, and so on.

In our research group [12], we proposed a method to estimate the version of the source code contained in the repository. We used the similarity[11]based on the longest common subsequence as the source code similarity, and presented a version based on the assumption that the source code with the highest similarity is the reuse source. This method requires the repo that was reused as input and the repo from which it was reused, so it cannot be used if the repo source is unknown.

2.2 Search by Locality-sensitive hashes: In this study, we consider the problem of detecting reuse of source files as a kind of problem of finding source files that are similar to the files given as a query. locality-sensitive hashing (also referred to as LSH) is used to speed up the search for a large number of objects.LSH is an algorithm used for approximate nearest neighbour search and clustering. [7]

There are several applications for LSH. Manku et al. [14] proposed a method for detecting Web pages with almost overlapping contents in Web crawling.Das et al. [3] proposed a method for recommending articles based on user behaviour using cooperative filtering for Google News. Brinza et al. [1] used for clustering of loci in genome-wide association analysis. Jing et al. [10] use LSH for image retrieval.

LSH is also used in source code. Jiang et al., [9] et al. use a fast detection method for similar subtrees in the source code to detect clones, and use LSH to perform clustering on feature vectors of subtrees. Yamanaka et al. [21] proposed a method of type 4 function clone detection by clustering using TF-IDF method as features, but LSH is used for feature vector clustering to perform fast detection.

3.Proposed method:

In this study, we propose a method to search a source file given as a search query for a large number of source files whose content of source code is similar to a query. By using LSH, fast retrieval is possible.

In this method, we also present an estimate of similarity to the retrieved results. This estimate can be efficiently obtained, and for each source file included in the search results, it is possible to know the degree of similarity with the query individually, although it is an estimate.

First, the similarity between the source files used in this method is defined, and LSH is explained, then the proposed method and details are explained.

3.1 Definition of Similarity between Source Files:

The similarity between the source files used in this study is the similarity using n-gram and Jaccard coefficients. The advantage of using this similarity is that it is not overly influenced by sorting of function units. In addition, as with TF-IDF method, similarity is not affected except for 2 files for which similarity is obtained. The specific calculation method consists of the following steps.

1) Lexical splitting of source files: Remove the comment from the source file s, split it into lexical columns, and get Ts. By removing comments, it is possible to prevent the size of comments and the difference in comments from affecting the similarity .Do a similar operation on the source file t to obtain a lexical string Tt.
2) (2) Extracting n-grams from Lexical sequences: Convert a substring of length n from a lexical sequence Ts to a multiplex set Ms of n-grams.

   For example, if n = 3, the n-gram obtained for the element sequence e1; e2; e3; :::; em is ff($;$; e1), ($; e1; e2), (e1; e2; e3), (e2; e3; e4),..., (em
3) (3) Conversion of n-grams from multiple sets to sets: Convert Ss to a set by appending a number to each element of the multiple set Ms. The number to be added to each element is a value from 1 to k for a given element e, if k elements are included in the multiplex before conversion, each element is appended with a value from 1 to k. For example, if a multiplex ffa ,a,b,b,b, cgg consisting of elements a; b; c is converted to a set using this technique, the resulting set is f(1; a), (2; a), (1; b), (2; b), (3; b), and (1; c)G.

   In the same way, we obtain the set St from the multiplex set Mt. However, the number to be added does not depend on Ms, and even if Ms and Mt contain the same elements, the number to be added in the process of converting from Mt to St is given in order from 1.

(4) Calculation of Jaccard coefficients for Sets: The similarity of the source file s; t is calculated by the following equation using the set Ss and St

Jaccard(Ss; St) $=\frac{|Ss \cap St|}{|Ss \cup St|}$

3.2 Locality-Sensitive Hashing:

locality-sensitive hashing is one of the methods used for similarity search. [7]In this study, LSH is constructed by using a family of hash functions F in which the following equation holds. [16]

$Pr_{h \in F}[h(x) = h(y)] = sin(x; y)$

where sin(x; y)is the degree of similarity between x and y, and 0 sin (x; y) is the degree of similarity between x and y.

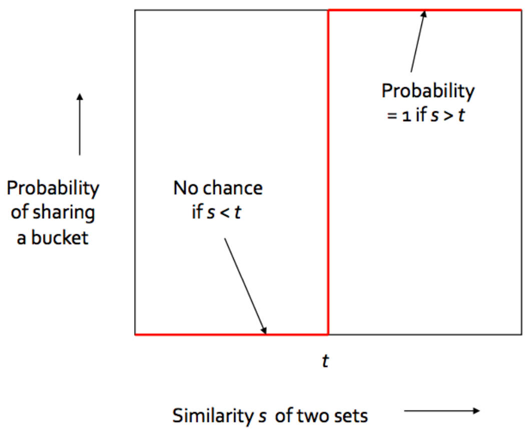


Figure 1. Probability of entering the same bucket

MinHash is a method for fast estimation of similarity between sets. Find the hash value for each element in the set and find the minimum value in it. When a similar operation is performed on another set, the probability that these 2 minimums coincide is equal to the Jaccard coefficient for the 2 sets.

LSH is constructed using the method described in literature[13].2 Think about sets. We prepare r b hash functions and find b vectors of MinHash values of r dimension for each set.2 Compare the vectors corresponding to each hash function between sets. At this time, the probability that MinHash values match between 2 sets, i.e., if the Jaccard coefficient is p, the probability that 1 or more vectors among the b vectors match is 1. If MinHash values match between two sets, the probability that MinHash values match between two sets is 1. If MinHash values match between two sets, MinHash values match between two sets, MinHash values match between two sets, MinHash values match between two sets, MinHash values match between two sets, MinHash values match between two sets In Figure 1, when the value of the parameter is actually given $1-(1-p^r)^b$ Show the graph of. The parameters are b= 120, and for r, r= 4; r= 8; r= 16 3 ways .As shown in the figure, you can adjust the probability of entering the same bucket by adjusting the parameters.

3.3 Searching for Source Files using LSH:

The proposed method performs a search using LSH on the database. By using a database, it corresponds to a case where the number of files to be searched is large and all necessary data cannot be stored in main memory.

The method consists of 2 steps .In the registration step, multiple source files are entered and registered in the database .In the search step, a source file q is given as a query, and a set of similar files as search results, and an estimate of the similarity between each result file and the source file q is output.

3.3.1 Registering to the Database to be searched

In this step, the source file to be searched is input and registered in the database.

First, prepare R B hash functions to be used in MinHash, which constitutes LSH.R; B is the upper bound of the LSH search parameter r; b. The hash function to be prepared is a function from n-gram with a number attached to the hash value, and they are called hi. However, 1 i R B is. Next, the source file to be searched is registered in the database, and the MinHash value obtained from each hash function is also registered in the database for 1 source file. The function that obtains the value of MinHash from the source file using hi is called mi .For a source file f, let g be the numbered n-grams and G be the function from the source file to the set of numbered n-grams, then mi(f) = ming2g(f)hi(g).Using mi,m1(f), m2(f),... Register mr B (f) at the same time.

LSH is configured using the MinHash value registered, and an index is created to speed up the search .An index is created for each B-tuple with R MinHash values as a pair. That is, (m1 (f),m2 (f),... 1 set of indices I1, (mr+1(f), mr+2(f),... In this case, mR 2(f))is set to 1 and the index I2 is set, and the total B-set index is set.

3.3.2 Searching for Similar Source Files:

In this step, a source file similar to a given source file is searched from the database. To search, in addition to the source file to be queried, parameters 1 r R and 1 b B of LSH are given. Find the value of MinHash for a given source file, and perform a search based on that value.1 For i b, using index Ii,

S_i(q)={f ∣$\forall j \in [1,r].mj$+R X (i-1) (q)=m j+R X (i-1)(f)}

Find the Si that will be. The union of this set Si, i.e. $U_{1\leq i\leq b}$Si(q) , is the result of the search, that is, the set of source files similar to the query.

In addition, for each source file included in the search results, an estimate of the similarity with the query is output. The similarity is estimated using maximum likelihood estimation.

Suppose that a similar source file in the search results is contained in the set S1; S2; :::; Sb of x sets. When similarity is expressed by p, x vectors for vectors of r dimension match, and b

$$L(p)=(p^r)^x(1-p^r)^{b-x}\binom{b}{x}$$

It is represented by. Therefore, given that the maximum likelihood estimator^p for similarity p is 0 p 1,,

$$P=\sqrt[r]{x \div b}$$

This is the first time I've ever seen this. As shown here, the possible values for the similarity estimate are limited to b in the case of x 1.In the case of r = 1, the interval of possible values as an estimate is equal, and it is a method itself to estimate the Jaccard coefficient using MinHash. On the other hand, for r > 1, the interval between possible values of the estimate becomes narrower as it approaches 1, finer as it approaches 1, and coarser as it approaches 0.

The bias of the estimator B(^p), variance V ar(^p), mean squared error M SE(^p)are

Equations2

In an example, the mean square error of the estimator when the parameter is b = 120 and r = 2, r = 4, and r = 8, while changing the value of p. The horizontal axis is the value of p, and the vertical axis is the mean squared error .It is shown that the mean squared error approaches 0 when the value of p approaches 1.

4.Implementation

4.1 About Similarity

The target of the method is the source file of C language, and n= 3 is the value of n of n-gram used for similarity .

Similarity between different files and different bars of the same file

Table 1Dataset

Package	**Version 1**	**Version 2**
original-awk	2012-12-20	2011-08-10
mgcv	1.8-4	1.7-12
seaview	4.3.1	4.5.3.1
fglrx-installer	8.96.	15.200
fglrx-installer-updates	1502..	8.960
Foreign	0.8.62	0.8.48
libhdhomerun	20140604	20120128
r-cran-eco	3.1-4	3.1-6
r-cran-xml	3.6-2	3.98-1.1
rt-tests	0.89	0.83

The difference in similarity between the two versions was investigated. The survey focused on the source code, 10 packages, and 505 files available in Ubuntu's apt. Table 1 shows 2 versions of orig for each package.tar.The source code obtained from gz was used.

The result of obtaining similarity among all the combinations of the target file is shown as the frequency for similarity in Figure 3.The figure on the left shows the similarity between files with different packages and different file names, and the figure on the right shows the similarity between files with different tar files, that is, different versions, with the same package, the same path, and the same file name. The left figure contains 1,13735 similarities, and the right figure contains 245 similarities. However, for 101 cases in the right figure, the similarity was 1.0.From this result, the similarity using 3-gram is considered to be effective in determining whether two files are different versions of the same file.

4.2 About Hash Functions

In the proposed method, it is necessary to prepare multiple hash functions for n-grams with added numbers. This time, we implemented the following as a simple method.

The hash value by the hash function hi for the numbered 3-gram(num; t1; t2; t3)is

$(((\text{num} \times 65537)+t_1) \times 65537+t_2) \times 65537+t_3) \times a_i + b_i$

Ask by. where num is the number appended to the 3-gram.

Where t1; t2; t3 are the values of Java's hash Code method for n-gram elements, that is, lexical, ai; bi are constants for each hash function. The operation is performed in 64bit, and the lower 64bit is left for the overflow during the operation.ai uses a 64-bit random number and a bit or of 1, and bi uses a 64-bit random number.

5.Case Study

5.1 Evaluating Search Results and Similarity Estimates:

We tested libpng and libcurl to confirm that high similarity to the query appears in the results of the search and low similarity does not appear in the results of the actual search.

5.1.1libpng of png.c

The experiment was carried out using all revisions of c and h files contained in libpng repository*3 as data sets. Specifically, from the list obtained by git rev-list --all --objects,.c files and.h targeted files. The target file count is 20977.A commit file png that is included in libpng and tagged with v1.6.0.I did a search with c as a query. The parameters of the search are r= 8; b= 120.

The number of files displayed in the search results is 716, which is 3.4% of the total data set .Among those that did not appear in the search results, the highest similarity value was 0.482.Also, the lowest similarity among those found in the search is 0.598.Figure 4 shows the relationship between the estimated value and the actual value of the similarity between the file and the query in the search results. The x-axis represents an estimate of the similarity, and the y-axis represents the actual similarity.

5.1.2 libcurl of url.c

Experiments were carried out using all the revisions of c and h files in libcurl's repository*4 as data sets. The target file count is 23273.The url of the commit that is included in libcurl and tagged with curl-7470.I did a search with c as a query.

The number of files displayed in the search results is 374 files, which is 1.6% of the total data set .Among those that did not appear in the search results, the highest similarity value was 0.578.The lowest similarity in the search results is 0.572.

5.2 Evaluating the Ability to Search for a File's Origin

A case study was conducted to identify the project, file, and version of the reuse source using an estimate of similarity. The subject of the case study is v8monkey*5

This is the source code of libpng included in the libpng.v8monkey records which version of libpng was updated in multiple commit messages, and changes have been made in multiple source code to support the Animated Portable Network Graphics format.

We prepared a repository of 200 projects including the libpng project as the target of the search. These projects were searched on GitHub in 2016/1/31 using the query "lib language:c", and the top 200 repositories of the search results were obtained on 2/1 the same year.

As an experimental procedure, first of all, in the project to be searched, all included in the history.c files and.I registered the h file in the database. The number of registered cases is 567113.Next, we list the v8monkey commits that indicate which version of libpng they have updated. A total of 14 commits were found to meet this requirement. The query was performed to find the files in modules/libimg / png among the files newly added or modified in those commits. However, the following files were excluded from the evaluation target.

- Source file consisting only of comments.
- mozpngconf.h. This file does not exist in libpng, and seems to have been added by the developers of v8monkey.

The target file is 18 files, a total of 197.The contents of 57 of them matched the source file, and the contents of the other 140 did not match the source file. The search parameters are: r= 8; b= 120

Table 2:Categorized results

	BenchMark 1		Total
	Meet	Not Satisfied	
Meet Criteria 2	177	10	187
Not Meeting Standard 2	5	5	10
Total	182	15	197

As a result of the search, the corresponding version of the file that was reused for all the results appeared in the search results. For 197 queries, the total number of files in the search results was 120001.The minimum number of files in search results for each query was 167, the maximum was 1031, and the average was 609.In addition to libpng files, the search results included a total of 3 projects using libpng (97 projects).
In addition, we investigated whether the estimate of similarity is the highest among the estimates of similarity in search results for each query. In order to understand the results, we classified the results into 2 categories from 4 criteria.
Criterion 1 Find an estimate of the similarity of the reuse source version
Criteria that will be the highest value of similarity estimates in the results 2 In the files in the search results, the version from which to reuse.
The true value of the similarity of the version is the highest value of the true value of the similarity that appears in the search results.
Criterion 2 is limited to files that appear in search results because it is not possible to obtain the true value of similarity with 197 queries for all files in the database due to time constraints. The results of this classification are shown in Table 2.
Of the 197 queries, 182 met criterion 1, meaning that the estimate of similarity between the query and the recorded source of reuse was equal to the highest estimate of similarity in the search results .For queries that satisfy criterion 1, the number of files with the highest similarity estimate in the search results for each query was at least 1, 56, and 11 on average .In addition, we obtained the number of files with the highest similarity estimate in each search result of a query that satisfies criterion 1, and the number of duplicates removed by ignoring the differences in comments and formats. However, an md5 checksum was used to

detect duplicates. As a result, the number of files that were removed from duplicates was at least 1, the maximum 18, and the average 2.3.

For 15 queries that did not meet criteria 1, if the similarity estimates were arranged in descending order for each search result, the recorded version ranking was from 2nd to 14th, with an average of 5.3.

There were 197 queries that met criteria 2 out of 187.In the remaining 10 cases, the similarity between the source version and the query is not the highest value even if it is true. Therefore,

Table 3Source les as search queries

ID	File name	Vector Number of matches	Search results Number of cases	LOC	Size [B]
0	mozpngconf.h	0	0	525	27513
1	pngwrite.c	14842	503	1590	51254
2	pngwtran.c	26355	453	572	17279
3	pngrtran.c	41179	818	4296	147369

Of the 187 queries that meet criteria 2, 10 that do not meet criteria 1, 5.3% are likely to be affected by the use of an estimate instead of a true value for similarity.

5.3 Performance Evaluation

The performance of the proposed method is evaluated .The CPU of the execution environment is an Intel(R) Xeon(R) CPU E5-2620 with 2 processors, 64GB of memory, and Windows(R) 7 Professional. Also, all database and search query files are stored on the SSD. The database used for evaluation is the same as 5.2

5.3.1 Evaluation of Registrations

The database contains 200 projects with a total of 567113 files. For registration, I took all the files from the Git repository once, saved them in a directory, and then registered them in the database. The time it took to register all the retrieved files from the directory was 230 minutes.

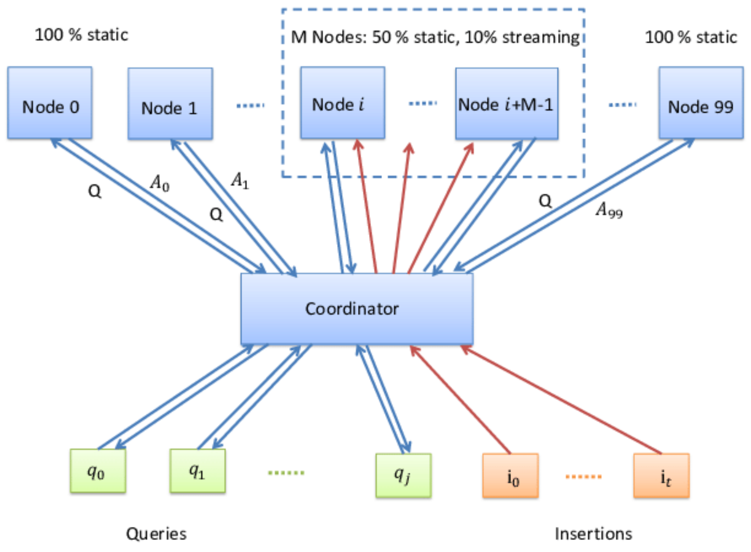
5.3.2 Evaluation of Searches

As a query for the search, v8monkey commit 3a04be0690dff135ec42784557fedbf6c572cd22 mod-ules/libimg/png I used 4 files contained below. Table 3 shows about them. Since it is expected that the longer the number of matching vectors in the database to be used for the search, the longer the execution time will be, so we selected from the queries used in 5.2 so that the number of matching vectors does not solidify.In addition, the number of matches in the vector is the largest among the queries used in 5.2.

Since the OS caches some of the database contents in memory when performing a search, the speed of subsequent queries is affected by the search of the preceding query. With this in mind, the order of searching for 4 files is all the way, i.e. 4! = 24 searches were performed in the same order, and the execution time was measured .The cache of the OS was cleared between each of the 24 searches, so that searches for the same file were not affected by the cache.

First, the time it takes to read the source file as a search query from disk and convert it to MinHash vector is summarized. As a result, the median values from ID0 to 3 were 47ms, 94ms, 63ms and 219ms, respectively.

It's a good idea. The horizontal axis represents the ID and the vertical axis represents the execution time in milliseconds. Where ID is the value divided by 4, the ID shown in Table 3, and the remainder indicates how many times the query was searched. For example, if the ID is a multiple of 4, it shows the result of the first search after clearing the cache, and if the remainder is 3, it shows the result of searching for the corresponding query after searching for other 3 queries.



6.Threat to validity

The implementation of this study uses a hash length of 64bit and assumes that the behaviour of the hash function behaves completely random, and does not consider the effect of the collision as the probability of collision of the hash value is very low. However, we have not verified whether the collision actually occurred or the effect on the result if the collision occurred .In addition, the bias of the hash function may have influenced the result of the case study.

Only 1 project was used as a query for the search in the case study. For this reason, the effectiveness of applying this method to other projects or projects that use other languages may differ from the effectiveness confirmed in this case study.

5. In Criterion 2, which was established for classifying results in 2, due to time constraints, the target of obtaining similarity was limited to only files that appeared in the search results. However, there is a possibility that some files that do not appear in the search results have a similar degree to the query than the files that appear in the search results.

7.Conclusion

In this study, a large number of source code files were obtained for source code files given as search queries.In this paper, we propose a method to search for those whose source code content is similar to a query. By using locality-sensitive hashing, a fast search was realized. In the case study, we searched for a total of 197 files including the files included in another commit, and for all 197 of them, we were able to search for the reuse source recorded in the commit message .Among them, the recorded estimate of the similarity of the reuse source for 182 cases was the highest estimate in the search results .In addition, the search was completed within 1 second per 1 search time.

In the proposed method, it is necessary to perform a search by specifying 1 source file for the search .The future challenge is to give the source code of the entire project and identify reuse and non-reuse from among them .In addition, the proposed method uses only information about 1 file, but it can be expected to improve the accuracy of the method by giving multiple files as input and using those information together.

References:

[1]Brinza, D., Schultz, M., Tesler, G. and Bafna, V.:RAPID detection of gene-gene interactions in genome-wide association studies,Bioinformatics, Vol. 26, No. 22,pp. 2856{2862 (2010).

[2]Broder, A. Z.: On the resemblance and containment ofdocuments,Compression and Complexity of Sequences1997. Proceedings, pp. 21{29 (1997).

[3]Das, A. S., Datar, M., Garg, A. and Rajaram, S.: GoogleNews Personalization: Scalable Online Collaborative Fil-tering,Proceedings of the 16th International Conferenceon World Wide Web, WWW '07, New York, NY, USA,ACM, pp. 271{280 (2007).

[4]Davies, J., German, D. M., Godfrey, M. W. and Hindle,A.: Software Bertillonage: Finding the Provenance of anEntity,Proceedings of the 8th Working Conference onMining Software Repositories, pp. 183{192 (2011).

[5]Davies, J., German, D. M., Godfrey, M. W. and Hindle,A.: Software Bertillonage: Determining the provenanceof software development artifacts,Empirical SoftwareEngineering, Vol. 18, pp. 1195{1237 (2013).

[6]Heinemann, L., Deissenboeck, F., Gleirscher, M., Hum-mel, B. and Irlbeck, M.: On the Extent and Nature ofSoftware Reuse in Open Source Java Projects,Proceed-ings of the 12th International Conference on SoftwareReuse, Lecture Notes in Computer Science, Vol. 6727,pp. 207{222 (2011).

[7]Indyk, P. and Motwani, R.: Approximate Nearest Neigh-bors: Towards Removing the Curse of Dimensionality,Proceedings of the Thirtieth Annual ACM Symposiumon Theory of Computing, STOC '98, New York, NY,USA, ACM, pp. 604{613 (1998).

[8]Inoue, K., Sasaki, Y., Xia, P. and Manabe, Y.: Wheredoes this code come from and where does it go? { In-tegrated code history tracker for open source systems{,Proceedings of the 34th International Conference onSoftware Engineering, pp. 331{341 (2012).

[9]Jiang, L., Misherghi, G., Su, Z. and Glondu, S.:DECKARD: Scalable and Accurate Tree-Based Detec-tion of Code Clones,Proceedings of the 29th Interna-tional Conference on Software Engineering, ICSE '07,Washington, DC, USA, IEEE Computer Society, pp. 96{105 (2007).

[10]Jing, Y. and Baluja, S.: VisualRank: Applying PageR-ank to Large-Scale Image Search,Pattern Analysis andMachine Intelligence, IEEE Transactions on, Vol. 30,No. 11, pp. 1877{1890 (2008).

[11]Kanda, T., Ishio, T. and Inoue, K.: Extraction of prod-uct evolution tree from source code of product variants,Proceedings of the 17th International Software Prod-uct Line Conference, Tokyo, Japan, ACM, pp. 141{150(2013).

[12]Kawamitsu, N., Ishio, T., Kanda, T., Kula, R. G., DeRoover, C. and Inoue, K.: Identifying Source Code Reuseacross Repositories using LCS-based Source Code Sim-ilarity,Proceedings of the 14th International WorkingConference on Source Code Analysis and Manipulation,pp. 305{314 (2014).

[13]Leskovec, J., Rajaraman, A. and Ullman, J. D.:Miningof Massive Datasets, chapter 3, Cambridge UniversityPress (2014).

[14]Manku, G. S., Jain, A. and Das Sarma, A.: DetectingNear-duplicates for Web Crawling,Proceedings of the16th International Conference on World Wide Web,WWW '07, New York, NY, USA, ACM, pp. 141{150(2007).

[15]Mohagheghi, P., Conradi, R., Killi, O. M. and Schwarz,H.: An empirical study of software reuse vs. defect-density and stability,Proceedings of the 26th Interna-tional Conference on Software Engineering, pp. 282{291(2004).

[16]Moses S., C.: Similarity Estimation Techniques fromRounding Algorithms,Proceedings of the Thiry-fourthAnnual ACM Symposium on Theory of Computing,STOC '02, New York, NY, USA, ACM, pp. 380{388(2002).

[17]Qiu, J., Su, X. and Ma, P.: Library Functions Identi ca-tion in Binary Code by Using Graph Isomorphism Test-ings,Proceedings of the 22nd IEEE International Con-ference on Software Analysis, Evolution, and Reengi-neering, pp. 261{270 (2015).

[18]Rubin, J., Czarnecki, K. and Chechik, M.: ManagingCloned Variants: A Framework and Experience,Pro-ceedings of the 17th International Software ProductLine Conference, pp. 101{110 (2013).

[19]Sbjrnsen, A., Willcock, J., Panas, T., Quinlan, D. andSu, Z.: Detecting Code Clones in Binary Executables,Proceedings of the 18th ACM International Symposiumon Software Testing and Analysis, ACM, pp. 117{128(2009).

[20]Xia, P., Matsushita, M., Yoshida, N. and Inoue, K.:Studying Reuse of Out-dated Third-party Code in OpenSource Projects,JSSST Computer Software, Vol. 30,No. 4, pp. 98{104 (2013).

[21] Hiroki Yamanaka, On-ro Choi, Norihiro Yoshida, Katsuo Inoue: Fast Functional Clone Detection based on Information Retrieval Technology, IPSJ, Vol. 55, No. 10, pp. 2245{2255 (2014)