Vol.2011-SE-174 No.2
2011/11/1

Development of a code clone search tool for open source repositories

Pei Xia†, Yuki Manabe†, Norihiro Yoshida††, Katsuro Inoue†

Finding code clones in the open source systems is one of important and demanding features for efficient and safe reuse of existing open source software. In this paper, we propose a novel search model, open code clone search, to explore code clones in open source repositories on the Internet. Based on this search model, we have designed and implemented a prototype system named *Open CCFinder*. This system takes a query code fragment as its input, and returns the code fragments containing the code clones with the query. It utilizes publicly available code search engines as external resources. Using *Open CCFinder*, We have conducted various case studies for Java code. These experiments show the applicability of our system.

1. Introduction

Open source repositories are growing fast. Millions of projects have been hosted in Open source repositories such as Google code, SourceForge.net, GitHub. They are playing very important roles in software development today. Even software systems in industry are increasingly using the open source systems due to their reliability and cost benefits [1].

One of usages of the open source repositories is to reuse existing source code for new systems. We can easily get source code files of various projects that hosted in open source repositories on the Internet.

When reusing source codes, some problems about software compliance may happen.

- When we find a useful source code file, can we reuse it safely?
- Are our own open source projects illegally reused by other people?

These questions are important, but difficult to answer. For the first question, before reusing the source code, developers should make sure that they will not violate the license. A license violation may take them to court and cost them a lot. However, to tell the license of a source file is not easy, because there are many code clones among open source projects [2]. That means they also copy and modify source code from other projects. In extremely cases they even change or remove the original license statement in the source files [3]. Reusing such source files is risky. For the second question, even though some other projects had reused source code while violating its license, the original code owner would hardly know it, since it is hard to check other projects by hand.

In order to answer such questions, one solution is to find out all the cloned code in the world, and compare the related information about them. Then we would be able to tell the reuse relationship between those codes. In these days, various kinds of code clone detection methods have been devised, and a lot of practical code clone detection tools have been developed and used [4], [5].

Base on the code clone detection technology, we proposed a novel approach for open source code clone search, and also implemented a prototype tool named *Open CCFinder*. *Open CCFinder* takes a code fragment as its query input, and returns a list of files from open source repositories that contain cloned code with the queried one, along with extra information such as license, copyright, last modified time and so on. This tool can support us to study the raised problems.

Using *Open CCFinder*, we have performed some case studies of source code exploration. One is to identify a Java code file *Base64.java* [6] which is a public domain code file that widely reused by other open source projects. Another case study is to search and collect clone information for all the source files in SSHTools projects, which is a popular JAVA SSH application [7].

In this paper, we first describe the overview of *Open CCFinder* including architecture and search process in Section 2. Then we introduce tool feathers in Section 3. Section 4 shows our case studies. Section 5 discusses our approach and Section 6 shows the related works. In Section 7, we conclude our discussions with some future works.

2. Overview of *Open CCFinder*

2.1 Architecture

Figure 1 shows the architecture of *Open CCFinder*. It takes an input query Q and returns an output results set R. Input query Q is composed of code fragment q_c and code attribute q_a. q_c may be a complete source file or a part of a source code file, which is in question. q_a is a set of associated information characterizing q_c, such as the file name. q_a is optional and could be added to improve the quality of the output results.

Given an input Query Q, *Open CCFinder* extracts useful information from it and generates queries for external code search engine (e.g. Google code search, SPARS etc.). And then analyze the returned candidate files from external search engines, at last form a final result as output R. The detail of this search process will be introduced in section 2.2.

† Graduate School of Information Science and Technology, Osaka University
†† Graduate School of Information Science, Nara Institute of Science and Technology

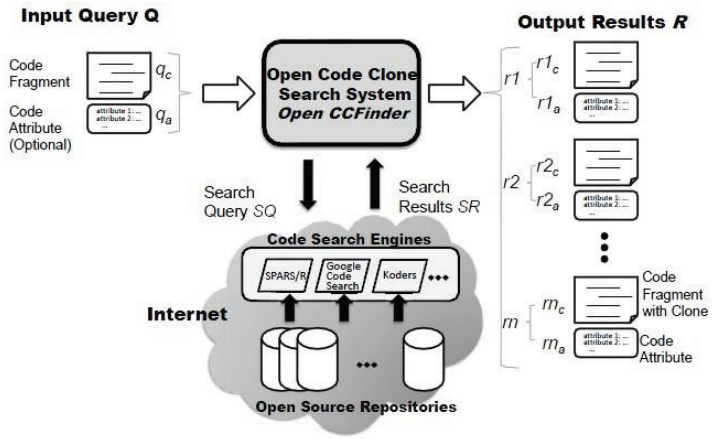

Figure 1 Architecture of *Open CCFinder*

Output result *R* is composed of results *r1, r2... rn*. Each result *ri* is composed of a code file ri_c and its code attribute ri_a. ri_c is code file that return by external code search engines, and ri_a a set of associated information about ri_c, including URL, file path, LOC, license, copyright, last modified time, clone cover ratio, shown as Table 1.

In addition, Figure 2 shows how we calculate cover ratio of a result file.

For the external code search engines, we use Google code search and SPARS in our tool implementation. Google code search is a famous code search engine, it provides search

Table 1 Attributes of output results

URL	where ri_c can be accessed on the Internet
File path	the file path of ri_c in its project
LOC	line of code of ri_c
License	the software license of the source file
Copyright	the copyright of the source file
Last modified time	the latest committed time of ri_c in its repository
Cover ratio	the code percentage of the queried code q_c that reused by ri_c

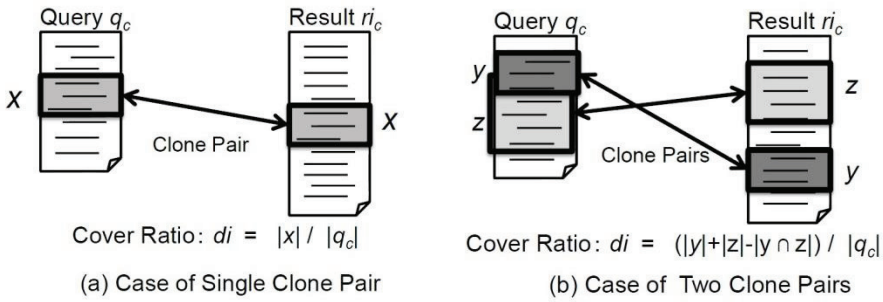

Figure 2 Definition of Cover Ratio

service API to user, so we can easily integrate it to our tool; SPARS is a Java component search engine with the keyword input and component rank mechanism developed by our research group [8]. The Java class repository of SPARS is kept updated by us.

2.2 Search process of *Open CCFinder*

(1) Word Extraction

At the beginning, code fragment q_c in input query Q is tokenized, the words from source code and comments are separated. Camel Case (e.g. helloWorld) or Snake Case (e.g. hello_world) words will not be decomposed into multiple words. User can choose to extract words from source code or from comments, or from both.

(2) Keyword ranking

Next, the keywords used for query generation are selected from the extracted words. In this step, first *Open CCFinder* filters out the words that considered being featureless. For example, the reserved words of each source code language, the words in very short length, and the words included in customized filter are filtered out. After the filtering, a simple words importance ranking strategy is applied on the remaining words. Currently there are two strategy implemented in the tool for ranking the words: frequency strategy and random strategy. Frequency strategy is to rank the keywords by the times they appear in the source codes or comment, while random strategy is just to rank the words randomly.

(3) Searching for candidates files

Using the ranked keywords, a search query SQ for the code search engines is created. As the search engines, we use SPARS/R which is Java class file search engine built by our research group, and Google Code Search whose API is well known of availability and flexibility. Both of the two search engines accept keywords sequence as their query input, so we use the combination of most frequently used words as SQ. If user wants, the additional

```
GetAppropriateCadidateFiles (keywordsList)
    1.  CadidateFiles = ϕ;
    2.  CurrentKeywords = ϕ;
    3.  While CadidateFiles is not appropriate   //Judged by user
    4.      ParticalCadidateFiles = ϕ;
    5.      While ParticalCadidateFiles is empty or too large size
    6.          CurrentKeywords = CurrentKeywords ∪ keywordsList.nextTopKeyword
    7.          ParticalCadidateFiles =   results searched with CurrentKeywords ;
    8.      End while
    9.      CurrentKeywords = ϕ;
    10.     Cadidates = Cadidates ∪ ParticalCadidateFiles
    11. End while
    12. Return Cadidates
```

Figure 3 The pseudo code of searching for candidate files with ranked keywords

input attribute file name also can be given to the search engines.

Then we generate several queries for each search engine to get appropriate candidate files. For each query, the returned results set from search engines should not be very large, for fear of including too many irrelevant results. When the returned results set are too large, we will add one more keyword from the ranked keywords list to the query to narrow the results set. At last we merge the returned results of several queries as the analysis candidate files.

The detail process is shown in Figure 3.

(4) **Downloading Candidate files**

All the candidate files in step 3 are downloaded from Internet. While downloading the file, the tool is also crawling the web to extract useful information for the code attributes such as file path, URL, LOC, License, Copyrights, and last modified time if available.

(5) **Code Clone analysis.**

The code clones between the input query code fragment q_c and each source code sr_i obtained at Step 4 are computed. We have used a code clone detection tool CCFinder [9], with its parameter setting for the minimum token length 15. We calculate the cloned code cover ratio of q_c for each Candidates.

(6) **Result Forming**

All the Candidate files and their code attributes are combined and packed as the output result R of this system, sorting by their cloned code cover ratio.

3. Implementation of *Open CCFinder*

Open CCFinder is implemented with Java language, which is platform independent. We also provide a friendly GUI tool using Java swing components for *Open CCFinder*. Here we mainly introduce it with this GUI tool. Besides, we also provide a command line tool for *Open CCFinder*.

3.1 Configuration

First, we have to configure the basic parameters for this tool.

(1) **Tokenize target**

User can choose to extract keywords from code or from comments content, or from the both. Default set is extracting from only source code while ignoring the comments content.

(2) **Keyword filter**

Keyword filter contains the words that considered being less important, like reserved words and featureless words such as “temp”, “str” etc. User can add or remove words from the filter.

(3) **Minimum token length**

This configuration is used to filter out short length words such as “id”, “i” that also be considered featureless. User can define the minimum length of tokens that used as keywords.

(4) **Keyword sort strategy**

Choose the strategy of ranking extracted keywords. *Frequency* means to sort keywords by the times they appear in the content. *Random* means just ranking them randomly.

(5) **External search engine**

Choose external search engines to use. Currently we only support Google code search and SPARS-J.

(6) **Maximum result number**

This is the limitation of results number returned from search engines for each query. Too large sized result set usually contains many irrelevant results. When the number of returned results is larger than this limitation, the tool will add one more high ranking keyword to current query and search more strictly to get a narrower result set.

(7) **Maximum keyword number**

This limits the maximum keyword number used for one query. If the tool cannot get an appropriate result set size with a query that had used maximum number of keywords, then no more keyword would be added, and this query is force to over, only a part of the results are returned.

(8) **Language**

Choose file types for searching. Currently *Open CCFinder* only supports Java, C++ and C.

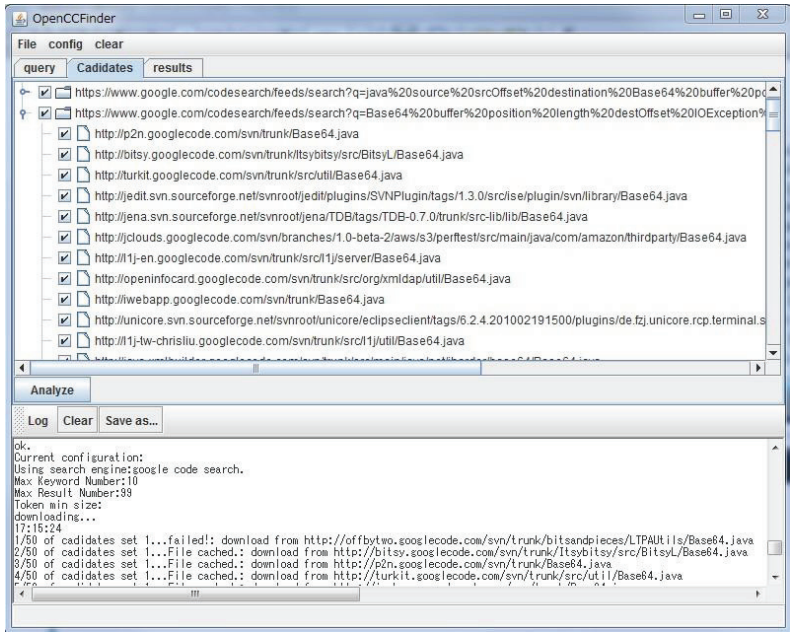

Figure 4 Candidates View

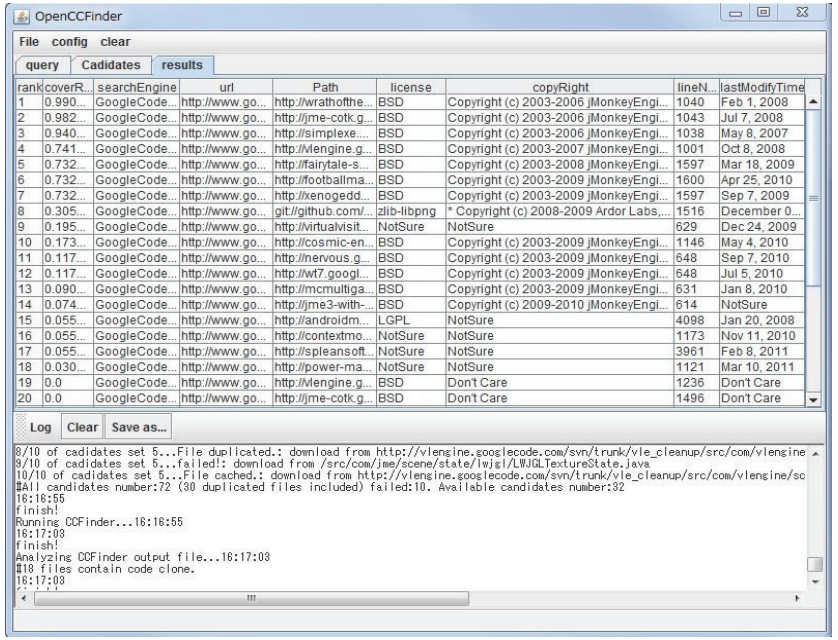

Figure 5 Results View

3.2 Searching

Users can input the code fragments they interested in into the query text area, or they can choose a local file directly using a file chooser. Then the tool would extract and rank the keywords of the input code fragment automatically. User can also adjust the keyword ranking by hand. After keywords are decided and "search" button are pulled, the tool will start searching with high ranked keywords in current keywords list and then return a partial candidate files set.

User can repeat this searching process using different keyword combination and get several partial candidate files sets, and then merge the partial candidate files together to get a large result sets to improve the recall, as shown in Figure 4.

In figure 4, each package icon represents a partial candidate files set. And each file icon represents a returned file. Use can then select the ones they would like to analyze by checking the check boxes on left side. When "Analyze" button was pulled, *Open CCFinder* begins to download the selected files. However, Some of the URLs returned by Google code search API are repository URLs that start with "svn://", "git://" etc., which cannot be downloaded through http request directly. We have to find patterns to translate the repository address to http URLs to get the raw file. Even though, a few files cannot be downloaded successfully, then *Open CCFinder* just tags it as "failed" and skips this file.

While downloading, *Open CCFinder* is also crawling the web to collect information about license and last modified time for each file. When all the files have been downloaded, code clone detection is applied on these downloaded files along with the input query code, so clone cover ratio for each file can be calculated.

3.3 Result

Final results are packed and shown in results view, as Figure 5.

In this view, each of the results is shown in one row, sorted by code clone cover ratio. High ranked results indicate that this file has more possibility of reusing/being reused by queried code fragment. Related information for each file including rank, external search engine, URL, file path, license, copyright, line of code, cover ratio and last modified time are shown in different column.

When one row is double clicked, the source code of the file will be displayed in a text area with pop-up dialog, from where user can view the detail content of the result file. And the cloned source code fragments of the file are colored.

In the implementation, we notice that not all the information can be searched out successfully. There are several “not sure” results in the license, copyright and last modified time columns.

The bottom panel in the tool displays the log information. It records keywords list, downloading state, time cost for each search. It helps user to understand the results better.

4. Case study

We have conducted two case studies to explore applicability of *Open CCFinder* approach. All these experiments of *Open CCFinder* have been performed under PC Workstation with dual Xeon X5550 2.66GHz processors and 24 GB memory between Aug. 2011 and Sep.2011.

4.1 Case study 1: *base64.java*

Case study 1 is designed for the first raised question: *When we find some useful source code, can we reuse it safely?*

Consider such a situation that we have found a file named *base64.java* in Apache ObjectRelationBridge (Apache OJB) open source project [10] and we would like to reuse it. The comments section in the source file represents that this file is under the Apache license. But we wondered if this file is copied from somewhere else that may be under another license. Then we take this *base64.java* file as input and search for similar files from open source repositories using *Open CCFinder*.

Open CCFinder returns 57 other files from open source repositories that contain code clone with the *base64.java*. For the limited space here, we cannot present all the detailed data of the 57 files. Instead we organized the data and draw a scatterplot view, as shown in Figure 6.

In the figure, the files are distributed by their cover ratio and last modified time. Licenses are shown in different icon. From it we can observe the following:

- 55 source files from other projects contain code clones of *base64.java*. The last modified time is varied from 2004 to current. The earliest file we can find in *Open CCFinder* is under Apache license.
- In these files, the cover ratios are not the same, which may indicate these files reused and modified from each other in different ways.
- Most of the licenses are found as not sure, while several files have been found under MIT, LGPL, GPL, BSD, Apache or AGPL licenses.

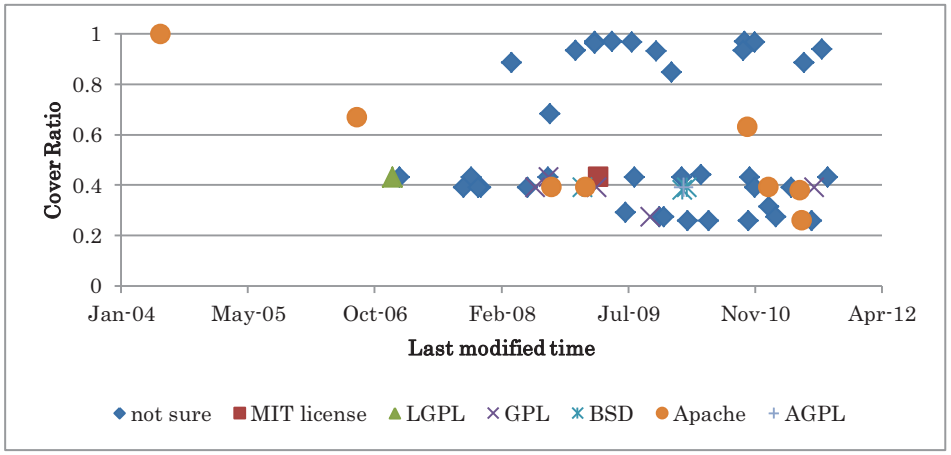

Figure 6 Files that contain code clones with *base64.java* found by *Open CCFinder*

The “not sure” license means those files are under some licenses that *Open CCFinder* cannot recognize. So we had to double click them in *Open CCFinder* and look into the comments statement in source code. At last, we found that most of the “not sure” licenses are described as “Public domain”. With further investigation by hand, we found a public domain project named Base64 on the internet, which supposed to be the original one. With such information, we are more confident to reuse the *base64.java* file of Apache OJB project. This file is supposed to be a modified version from the public domain project base64 and legally change it to the Apache license. So keep this file as Apache license is OK. Another choice is to reuse the public domain code, and then we can change the file to some other license legally.

Though *Open CCFinder* cannot tell the answer of the first question directly, it helps us to do the study easier.

4.2 Case study 2

Case study 2 is designed for the second question: *Are our own open source projects illegally reused by other people?* In this case study, we investigate a Java project SSHTools, try to find some files that illegally reused by other projects.

SSHTools is a Java SSH application providing Java SSH API, terminal and so on, which is under the GPL license. We choose this project because it is widely used by other projects, and its small project size which is easy to analyze.

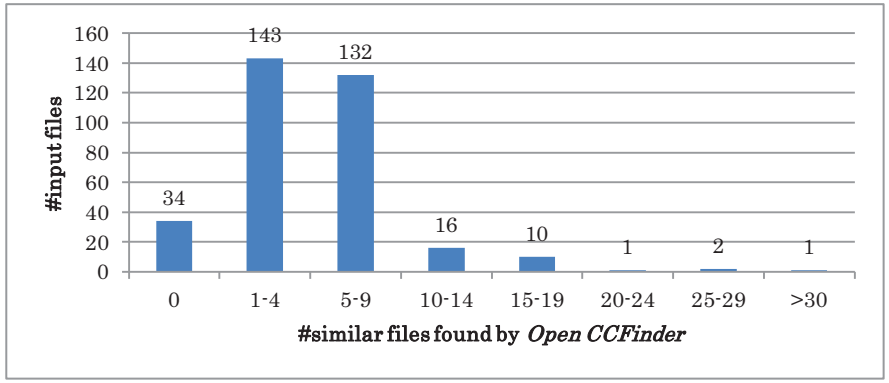

Figure 7 The histogram of #input files in terms of #similar files found by *Open CCFinder*

Ignoring some tiny sized files, we have selected 339 files from SSHTools project (version 0.2.9, last modified time is 6-23-2007) as input files, and used the command tool of *Open CCFinder* to analyses them. We counted the number of similar files found in open source repositories for each SSHTools' file, and also count the number of different license of similar file. In this case study, we set up a threshold on cover ratio to filter out the similar files. Only those files which cover more than 40% code of the queried file would be counted as similar files. The result is shown as Figure 7. The figure shows the number of similar files found in each of the 339 SSHTools' files.

From this figure, we can observe that 305 of the 339 files contain code clones with other files from open source repositories. 275 of them have less than 10 similar files found for each; several files have 10-30 similar files found for each; one file has more than 30 similar files.

Besides, we also investigated the different licenses appeared in each similar file. SSHTools is under GPL license. However, 285 files in SSHTools have similar files found with 1 different license, 10 files in SSHTools have similar files with 2 different licenses, and 1 file in SSHTools have similar files with 3 different licenses.

We checked the detail of the data and found another project reusing SSHTools as a third-party component, which named i-service project, stored in Google code repository is under LGPL license. These two projects are both found out by *Open CCFinder* for most of the searches. So it is not strange that there are many cloned files under 1 different license besides GPL license. Those LGPL are almost from i-service project. We also checked the files whose cloned files have 2 or 3 different licenses. There are several unusual source code reuse cases. Here we only state one of them.

Table 2 The case that similar files are under 3 other license (partial results)

file path	Project name	Cover ratio	license	Last modified
/j2ssh-fork/src/com/sshtools/common/util/BrowserLauncher.java	j2ssh-fork	0.91	GPL	2008/6/17
/de.fzj.unicore.rcp.terminal.ssh.gsissh/.../sshtools/common/util/BrowserLauncher.java	unicore	0.89	LGPL	2010/2/3
/openfire/launcher/BrowserLauncher.java	openfire-tomcat	0.88	Apache	2010/4/19
/dg/hipster/BrowserLauncher.java	hipster	0.84	BSD	2006/10/12
…	…	…	…	…

Table 2 shows the similar files returned by *Open CCFinder* that 3 other different licenses exist, along with extra information about file path, project name, cover ratio and last modified time. For the space limitation, we only present a small part of the results here.

From this table, we can see the 4 files named *BrowserLauncher.java* with very high cover ratio. It is reasonable for us to suppose some of them have been reused by each other. Beside the GPL license, there are LGPL, Apache and BSD licensed similar files exist. But some of the license changes should not happen. For example, to change a GPL license file to Apache may cause legality problems.

This case study shows that *Open CCFinder* is helpful for answering the second raised question. We can find candidates of the suspicious files easily and effectively by using this tool.

5. Discussion

5.1 The availability of *Open CCFinder*

As shown in case study, *Open CCFinder* is helpful to analyze how source code is reuse. In case study 1, from the open source repositories we search out many similar files of Apache OJB's file *base64.java*. With extra information about each file, we can know how the searched code is used in different projects. Then developers' reuse activity that we focus on become easy and efficient. In case study 2, we analyzed a small java open source system named SSHTools. Finally we search out thousands of other files from open source repositories contains code clones with SSHTools' files, among which there are several unusual cases that may be illegal. Case study 2 also showed the availability of *Open CCFinder*.

However, this tool only provides some clues to get evidence, the final judgment on the legality issue should be made by human after all.

5.2 Performance

It takes about 1-2 minutes for *Open CCFinder* to complete one search task, including keyword extracting, downloading, collecting information and running CCFinder. Most of the time is spend at downloading step. Network traffic and the size of file to be downloaded also affect the executing time. Currently the tool is implemented as a single thread program, therefor all the source files are downloaded sequentially. In the future we can improve it by using multiple download threads, if not violate the terms of use of target repositories. Anyway, the overall performance might be bounded by the performance of the code search engines and the network environment.

5.3 Recall and precision

Open CCFinder searches for code clones in open source repositories using external code search engines. So the recall and precision of this tool depends on those search engines. It is difficult to evaluate recall of *Open CCFinder* quantitatively, because we could never know all the files in open source repositories. In addition, the search process limits the recall. *Open CCFinder* has to download files from the web, but considering the physical size of the disk and searching performance, it cannot download all the files stored in open source repositories. In the extremely case that millions of cloned files existing in open source space, *Open CCFinder* would only download a small part of them and the recall would become very low.

The precision of *Open CCFinder* has been calculated. In this discussion, it is defined as the ratio of files containing code clones to all the downloaded files. In case study 1, 62 files has been downloaded, of which 55 files contain code clone with the queried file. The precision can be simply calculated as 0.887; in the case study 2, overall we have downloaded 17054 files from internet, and 2480 of the files have been detected as containing code clone with the files in SSHTools. So the average precision is 0.145. The precision in case study 2 is low.

Using filename as keywords can improve the precision a lot, but the recall will drop, for missing some cloned files with different filename. Recall and precision highly depends on the keyword select strategy.

5.4 Keyword ranking strategy

The most important question in keywords ranking is how to pick out the words that can express the characteristics of cloned code fragment powerfully. We have discussed a lot about it.

However, for ease of implementation, currently there are only two strategies for keyword ranking. One is to rank by frequency, another is to rank randomly. In the two case studies, we applied the frequency strategy. But the low precision of 0.145 in case study 2 indicates that this is not a good strategy. While randomly ranking sometimes get very good results, but not stable.

In future work, we will implement some other keyword ranking strategy, for example, we are about to apply the TF-IDF algorithm to build an incremental knowledge base in advance, and then rank our words based on this knowledge base. This strategy is supposed to work better.

6. Related work

6.1 Code clone detection

Code clone is one of very active research areas in software engineering [11]. There are many research publications on detection algorithms, tools, empirical analyses, and applications of code clones. Roy et. al. have summarized and categorized those researches very well [12]; thus, we do not list up each of those here. Those studies mostly focus on the efficiency, scalability, and accuracy of clone detection and analysis for the source programs inside local repositories. We are interested in clone detection for the open source collection in the Internet space.

6.2 Origin and Evolution of Code

There are many research studies on analyzing and tracing code origin, provenance, evolution, genealogy, and so on through code clone analysis [13],[14],[15],[16],[17],[18]. Duala-Ekoko et. al propose Clone Tracker to trace and manage code clone history [19]. They have developed a tool for supporting clone tracking, with abstract clone information named clone region descriptor. Davies et. al. propose Software Bertillonage for determining the origin of code entities with anchored signature matching method [20]. These researches are closely related to our work. However, their objectives are different from ours in the sense that they analyze various characteristics of code fragment in their local repositories. In our case, we analyze the query code in Internet repositories.

6.3 Code Search Engines

Code search is not only a very emerging research area but also a very useful resource for software engineers these days [21]. We have used Google Code Search and SPARS/R as the code search engines in our *Open CCFinder*. Google Code Search provides search features with keywords associated with optional attributes such as package names, languages, and licenses. SPARS/R allows only keywords as its input. There are various different code search engines with different types of query inputs and search mechanisms, but none of those provides the code search features with both the code fragment query input and the code clone detection.

7. Conclusion

In this paper, we have proposed a novel concept for open code clone search, and presented its search model and detailed processes for *Open CCFinder* which is a prototype system for the open code clone search. We have conducted experiments with two case studies, which show the applicability of our approach.

There are several future works. One is to improve the performance of the current prototype implementation of *Open CCFinder*. Another would be to implement a new algorithm of keyword extraction to get better recall and precision.

Acknowledgements This work was supported by KAKENHI (No.21240002, No.23650015).

Reference

1) C. Ebert (ed.), “Open Source Software in Industry”, IEEE Software, Vol. 25, No. 3, pp. 52-53, May/June 2008.
2) S. Livieri, Y. Higo, M. Matsushita, K. Inoue, “Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder”, Proc. of 29th International Conference on Software Engineering (ICSE 2007), pp.106-115, Minneapolis, MN, May 2007.
3) Arne, P.H. 2008. "Jacobsen v. Katzer - Open Source License Validation: How Far Does It Go?," The Computer & Internet Lawyer (25:11), pp 27-31.
4) J. Cordy, K. Inoue, R. Koschke, and S. Jarzabek (ed.), “4th International Workshop on Software Clones (IWSC 2010)”, Cape Town, South Africa, May 2010.
5) C. K. Roy, James R. Cordy, R. Koschke, “Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach”, Science of Computer Programming, Vol. 74, No. 7, pp. 470-495, 2009.
6) Base64: Public Domain Base64 Encoder/Decoder, http://iharder.sourceforge.net/current/java/base64/
7) SSHTools Source Repository, http://sourceforge.net/projects/sshtools/.
8) K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, “Ranking Significance of Software Components Based on Use Relations”, IEEE Trans. on Software Engineering, Vol. 31, No. 3, pp. 213-225, Mar. 2005.
9) T. Kamiya, S. Kusumoto, K. Inoue: “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code”, IEEE Trans. on Software Engineering, Vol. 28, No. 7, pp. 654-670, July 2002.
10) Apache e ObJectRelationalBridge – OJB, http://db.apache.org/ojb/
11) J. Cordy, K. Inoue, R. Koschke, and S. Jarzabek (ed.), “4th International Workshop on Software Clones (IWSC 2010)”, Cape Town, South Africa, May 2010.
12) C. K. Roy, James R. Cordy, R. Koschke, “Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach”, Science of Computer Programming, Vol. 74, No. 7, pp. 470-495, 2009.
13) M. Godfrey, and L. Zou, “Using Origin Analysis to Detect Merging and Splitting of Source Code Entities”, IEEE Tran. on Software Engineering, Vol. 31, No. 2, Feb. 2005.
14) C. Kapser, and M. W. Godfrey, “’Cloning considered harmful’ considered harmful: Patterns of cloning in software”, Empirical Software Engineering, Vol. 13, No. 6, pp. 645-692, 2008.
15) S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, “MUDABlue: An Automatic Categorization System for Open Source Repositories”, J. of Systems and Software Vol. 79, No. 7, pp.939-953, 2006.
16) M. Kim, V. Sazawal, D. Notkin, and G. Murphy, “An empirical study of code clone genealogies,” Proc. of Foundations of Software Engineering (ESEC/FSE 2005), Vol. 30, No. 5, pp. 187-196, Lisbon, Portugal, Sep. 2005.
17) A. Lozano, M. Wermelinger, B. Nuseibeh, “Evaluating the Harmfulness of Cloning: A Change Based Experiment”, Proc. of Mining Software Repositories (MSR 2007), p. 18-21, Minneapolis, MN, May 2007.
18) S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, “An empirical study on the maintenance of source code clones”, Empirical Software Engineering, Vol. 15, No. 1, pp. 1-34, 2009.
19) E. Duala-Ekoko, M. P. Robillard, “Clone Region Descriptors: Representing and Tracking Duplication in Source Code”, ACM Tran. on Software Engineering, Vol. 20, No. 1, Article 3, pp. 3.1-3.31, Jun. 2010.
20) J. Davies, D. M. German, and M. W. Godfrey, “Software Bertillonage: Finding the Provenance of an Entity”, Proc. of Working Conference on Mining Software Repositories (MSR 2011), pp. 183-192, Honolulu, Hawaii, May 2011.
21) S. Bajracharya, A. Kuhn, and Y. Ye (ed.), “Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation”, Cape Town, South Africa, May 2010.