Similarity Calculasion of Two Jar Files

Meng Yao

School of Computer science, Carleton University, 1125 Colonel By Drive, Ottawa, ON 6139V6, Canada

This paper describes a system which can determine the percentage of similarity between two Java Archive (JAR) files. The system is designed based on three kinds of information abstracted from Jar files-- the license and attributions, the name of all classes, the logical structure of each class. An assisting tool is used to encoding the class file into source code. The first two kinds of information is usually used as weight in the final calculating and percentage of similarity will then be presented based on the number of matched logical structure of class of those two Jar files. Several comments regarding to the detail of each kinds of information are also presented at the end of the execution of the system.

***Index Terms*—Open source license, fingerprint, encoding, fingerprint generator detector**

I. Introduction

A. Problem

As the development of open source software, it is convenient to acquire the source code under open source license. However, there are rules that have to be obeyed prior to use or modify or distribute the source code of such software for the purpose of profit or not. Those rules may be different for each OSS depending on the type of license that is being applied to that particular software, but the bottom line is the actual year of copyright and the correct name of the creator in the project.

B. Motivation

Due to the existing of the problem mentioned above, it is necessary to find out whether one Jar file violates the rules of certain software license by using the code of open source software without the proper declaration of the copyright. So, calculating the simulation of two Jar files is a basic support to solve the license violence problem. The word “similar” in this particular case means that if a piece of code exists on both JAR files. But we do not pay attention to other aspect of the violation of a specific open source license. What we focus is just the level of source code.

C. Goals

The general goal is to calculate the similarity as fast as possible. However, it can be a difficult and time consuming job to compare each code line in each source code file. Thus, the demand for a tool which can determine the similarity of two Jar files effectively and efficiently is high.

D. Objectives

This project designs two steps to realize the goals. First step is to generate a fingerprint for the Jar file. The second step is to generate the result from the fingerprint and a jar file, in which a new fingerprint will be generated from the jar file and the result is actually generated from the two fingerprints. In the process of comparison, there are three kinds of information used to generate the result: the license and attributions, the name of all classes, the logical structure of each classed.

E. Outline

Section 2 of this paper describes about the background of this project, section 3 is about approach taken to do the project, section 4 is the validations, section 5 is conclusion and section 6 is the list of references used in this paper.

II. Background

A. Introduction of open source and open source license

Open-source software (OSS) is computer software that is available in source code form for which the source code and certain other rights normally reserved for copyright holders are provided under a software license that permits users to study, change, and improve the software. Open source licenses meet the requirements of the Open Source Definition. Some open source software is available within the public domain. Open source software is very often developed in a public, collaborative manner. Open-source software is the most prominent example of open-source development and often compared to (technically defined) user-generated content or (legally defined) open content movements.[1] The term open-source software originated as part of a marketing campaign for free software.[2] A report by Standish Group states that adoption of open-source software models has resulted in savings of about $60 billion per year to consumers.[3][4]

An open source license is a copyright license for computer software that makes the source code available for everyone to use. This allows end users to review and modify the source code for their own customization and/or troubleshooting needs. Open source licenses are also commonly free, allowing for modification, redistribution, and commercial use without having to pay the original author. Some open source licenses only permit modification of the source code for personal use or only permit non-commercial redistribution. All such licenses usually have additional restrictions such as a requirement to preserve the name of the authors and a copyright statement within the code. One popular set of free open source software licenses are those approved by the Open Source Initiative (OSI) based on their Open Source Definition (OSD)..

B. *Assisting OSS used in the program*

This project use an open source application which can decompile a jar file into a ZIP file where the .class file are all replaced by .java file.

This application is supposed to run to decompile the jar file before running the project. The process is opening the jar file using the "jd-gui" application firstly, then click the menu of "File", then click the submenu of "save JAR scouces", then follow the instruction to save the zip file under the default name in the path as same as the Jar file.

III. DESIGN

The approach to compare two Jar files can be divided into two independent parts. The first is to generate the fingerprint of the Jar file which contains the essential and logical information of the Jar file. The second part is to compare the well formed two fingerprints in order to get the similarity of the two Jar files.

A. *The format of the fingerprint*

A fingerprint is an XML file which can explicates the core information of the Jar file from three aspects: the aspect of license, the aspect of name of classes and the aspect of logic structure of classes.

In the aspect of license, the program abstracts the name of the license that the Jar file is under and the attributions of the Jar file if it is an open Jar file. Because the number of open source licenses are too large (the number of licenses which has been approved by OSI is up to 66), so this program just consider the six most popular open source license, including MIT, BSD, GPL, LGPL, Apache1.1, Apache2.0, EPL. So, the element named “license” just have 6 kinds of context. As to the attribution information, each element named “attribution” include a pair of elements named “author” and “year” respectively, which indicate contributor and the time of the Jar file. The number of the element named “attribution” depends on the information abstracted from relative files. If the Jar file is not open source software or lost its license information, these elements would be empty.

In the aspect of the name of classes, the program acquires every class’s name and put each name in the tag of “classname”. So, there would be the same number of “tag” with the name of “classname” as the number of class in the Jar file.

In the aspect of the logical structure of classes, the program generate a string for each class to show its logical structure, which is composed by the key words, like “if”, “else”, “while” and “for” with a specific numerical prefix to show the different levels of each key word. The program put the logical structure of every class in the tag of “classlogic”. If a class does not have these key words, the corresponding element would be empty.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
  <fingerprintId>mengyao.carleton.ca</fingerprintId>
  <license>MIT</license>
  <attribution>
    <author>mengyao</author>
    <year>2010</year>
  </attribution>
  <attribution>
    <author>mengyao</author>
    <year>2009</year>
  </attribution>
  <classname>YMVIW/ACTIVATOR.CLASS</classname>
  <classname>YMVIW/VIEWS/SAMPLEVIEW$ADDBUTTONSELECTIONLISTENER.CLASS</classname>
  <classname>YMVIW/VIEWS/SAMPLEVIEW$DELETEALLBUTTONSELECTIONLISTENER.CLASS</classname>
  <classname>YMVIW/VIEWS/SAMPLEVIEW$DELETEBUTTONSELECTIONLISTENER.CLASS</classname>
  <classname>YMVIW/VIEWS/SAMPLEVIEW$TODOITEMTEXTMODIFYLISTENER.CLASS</classname>
  <classname>YMVIW/VIEWS/SAMPLEVIEW$TODOITEMSLISTSELECTIONLISTENER.CLASS</classname>
  <classname>YMVIW/VIEWS/SAMPLEVIEW.CLASS</classname>
  <classlogic/>
  <classlogic>1if1if1while</classlogic>
  <classlogic>2if2if2if2if2if2if2elseIfor1for1for</classlogic>
  <classlogic>1if2if1if</classlogic><classlogic/>
  <classlogic>1for1if</classlogic>
  <classlogic>1if1if1if2if1if1if1if1if1if1if1while2if</classlogic>
</root>
```

B. *The way to abstract essential information from the Jar file*

As to acquiring the information about the license of the Jar file, what the program does in the first step is to find the file which contains a copy of the license because the requirement --“a copy of this Agreement must be included with each copy of the Program” is a common requirement for every license. Although the name of the text file which contains a copy of the argument is not specified by each license, we can get the preferable name of the file which are recommended by some license, such as “LICENSE”, “COPYING”, ” README”. If we cannot find the file with these name, then every text file have the possibility to contain a copy of license. The next step is to compare the content of a test file to the content of six licenses one by one. If one license matched, then we got the license the Jar file is using. If no text file matches any license, then we give up finding the license information including the next step of finding the attribution of the Jar file. In the second step, what the program does is to get the attribution information of the Jar file by the instruction of the license if the name of the license has been got in the last step obviously. Learned from these agreements, each license has indicated the location and the format of the attribution information. For example, the Apache license specifies that:

“to apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) .The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives. “

“Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License. "

As to other license which did not specified so strictly, such as BSD, they still require the well formed attribution information -- “Copyright yyyy name of copyright owner” to be showed in source code. So the program can get the attribution information by the clue of the license. If the program did not match this information, the element “attribution” would be empty.

As to the name of classes, what the program did is just to acquire the name of class file, and then put the name in xml tag named “classname” in fingerprint.

As to the logical structure of the classes, what the program did is to abstract the key words and the relationship of these key words. Because the variable names and method names and the class names in a Jar file can be modified easily by existing tools, it is possible that these names are modified if the releaser wants to cover the origin of the source code. But what cannot be changed in a class file is the logic of the code which can be illustrated by these key words including “if” ”else” ”for” and ”while”. So, the program abstracts the logic of a class to a string. For example, the string “1if1else2if3if1for2for1while” means that in this class file, first, there is an IF block, then the ELSE block. And in the ELSE part there is a nested IF block in which there is another nested IF block. So, it is a three level loop in the beginning part of the class file. Then it comes a FOR loop nesting another FOR loop, then there is a WHILE loop in the last part.

C. The way to compare the two fingerprints

Because the target is to compare the two fingerprints as fast as possible, so the program do not go to compare the source code (logical structure of the class file) at the beginning. The first step in the comparison process is to match the license and the attributions, if the two fingerprints both have this kind of information and completely matched, which can lead to the result that the two fingerprints are 100 percent matched. Otherwise, the program generates the percentage of the matched part (from 0.1 to less than 1.0) and keeps it as a weight for calculating the final simulation percentage in the last step.

In the second step, the program goes to compare the content in classname tags one by one. If the classname of the two Jar file are completely matched, then the program will directly generate the result that the two fingerprints are 100 percent matched. Because it is possible that the two Jar file are the same file or the one Jar file is a part of the other one without any change. But they just lost their copyright notice. So there is no need to go to the next step to compare the two Jar file in the source code level if the classname of two Jar file are completely matched.

In the third step, the program goes to compare the content in the tag of “classlogic” which happened in most situations, because the first two steps can be seen as the rapid ways to solve the problem which just can be applied in special situation. The third step is the general way to compare two Jar file. The work in this step is to match the

In the fourth step, the program goes to calculate the percentage of the similarity of the two fingerprints. After the first steps, the program have got the percentage of the matched tag of attribution which is W (attribution), and the percentage of the matched classname which is W (classname), and the percentage of the matched logical structure which is W (logic). So, the final result is calculated by this way:

1. If W(attribution)=0 and W(classname)=0 and W(logic)>=0.5, similarity = W(logic)
2. If W(attribution)>0, similarity = W(logic);.
3. If W(attribution)=0 and W(classname)>0.5, similarity = (W(logic)+W(classname))/2
4. In other situation, similarity = W (logic)/2

Here are the explanations: if the two fingerprints have no common information about license and attribution and name of classes, the similarity of the two fingerprints is as half as the percentage of the matched logical structure. Because the logic structure is just a fatal and unchangeable attribute of a class file but is not a unique attribute. So the similarity of the logical structure does not necessarily mean the similarity and cannot be transferred into the similarity of the whole Jar files. So the program cut down half percentage of the similarity by the principle of Randomly assigned. But in case of more than half of the logical structure of the class files in Jar file are matched, it’s obviously that the two Jar file have a strong relationship between each other, so it is not necessary to cut down half of the similarity. Likewise, if the two Jar files have some attribution information in common, we can get to know that the two Jar files have a strong relationship between each other (maybe they are the some software in different versions). So the similarity of logical structure can exactly reflect the general simulation. As to the similarity of the name of the classname, it can be a weight to add to the general similarity if it is bigger than 0.5 which is large enough to reflect the relationship of the two Jar files.

D. The format of the result

The result not only gives the similarity of the two fingerprints but also give additional information to support customers. Firstly, the program offers the name of the licenses for each Jar file if they have any to help the customer make a more sensible decision through the detail of agreements. For example, one Jar file under the license of EPL and other Jar file under the license of GPL are not possible to share the same source code because the two licenses are not compatible which means the derived work from a software under the license of EPL is not supposed to released under GPL. So the real similarity may be not so high as in the result. Secondly,

Here is an example result to compare the same Jar file:

jar file name: org-netbeans-modules-classfile.jar
jar File name: org-netbeans-modules-classfile.jar
the CertaintyPercentage of the two Jar file is: 1.0
the former jarfile doesn't have a license
the latter jarfile doesn't have a license
the two jar has absolutely same classnames, there is no need to compare the source code!
The CertaintyPercentage is 100%

Here is an example result to compare two absolutely different Jar files:

jar File name: tomcat-dbcp.jar
jar File name: org-netbeans-modules-classfile.jar
the CertaintyPercentage of the two Jar file is: 0.17391305
the former jarfile is under the license of APACHEV20
the latter jarfile doesn't have a license
the similarity of the ATTRIBUTIONS of the two jarfile is 0.0
the similarity of the NAMES of CLASSES of the two jarfile is 0.0
the similarity of the LOGIC of CLASSES of the two jar is 0.3478261,waived of the 20 class files which has no loop statement or judgement statement in the total of 61 classes
so the overall certaintyPercentage is 0.17391305.

IV. VALIDATION

The result of the project is as the same as we expected. If we compare two totally different Jar files, the result is usually less than 0.2; If we compare the same Jar files, then result is 1.0 and it don't even compare the logic structure of each classes as the two Jar file have the same class names; if we compare a Jar file which is built by deleting some portion of the original Jar file with the fingerprint of original one, then the result is 1.0 but the result in the comment showed that how many classes in the Jar files are matched. As to the time-cost in the process of comparing, it cost about one seconds to generate two fingerprints and get the result of the comparison. To sum up, the result is sensible and useful for customer to judge the similarity of two jar files.

V. CONCLUSION

A. Advantages

Firstly, the time-cost of the project is rather low in general. And in some special case, such as the same two Jar files, it would perform better. Secondly, this project considers open source licenses, which make it more sensible when calculating the similarity and sometimes fasten the process of comparison. Thirdly, the program has good expansibility to add other popular license besides the six most popular open source licenses. The only job is just to add the detail of license in the way defined in the file named "ConstantPara.jave" in the project.

B. Future wok

Firstly, this program can be improved by modifying the assisting open source application –" jd-gui" and nesting this application into the program which can simplify the user action and make the project a unity. Secondly, the program can be improved by considering the inheritance relationship between classes, which could add as a weight to generate the result to make the result more reliable.