Fingerprinting Jar Files Using Winnowing and K-grams

Najmeh Taleb
Computer Science
Carleton University
ntaleb@connect.carleton.ca

Zeinab Bahmani
Computer Science
Carleton University
zbahmani@connect.carleton.ca

ABSTRACT
We applied the use of winnowing and k-grams to fingerprint jar files, and attempted to detect jars that contained or had significant similarities to other jars. Indeed, this paper presents software which can be used to check if a similarity exists between/among Java JAR files. When similarity happens, variable, constant, class, and method names could be changed, code sequences could be wrapped into different methods, and methods could be moved to different Java classes. The tool catches similarity by analyzing the Java bytecode sequences that describe the essential design/algorithm of software. Extracting finger print from Java opcode sequences are based on famous k-gram and Winnowing algorithms.

Keywords
k-grams, Winnowing, JAR file.

1. INTRODUCTION
The problems we have is how to detect software copies. Generally speaking, software systems often contain sections of code that are very similar. There are two main kinds of similarity between code fragments: similar based on the similarity of their program text; or similar based on their functionality. The first kind of similarity is often the result of copying and pastes certain code fragments.
our focus is on functionality aspects of the tools. We will try to detect and at the same time to minimize the false positive and false negative errors. The performance and space usage optimizations (if any) will be the future goals. Indeed, our objective is to have a tool which generates finger print from an original JAR file, and we use the finger print to detect if a suspected JAR file has cloned partially or wholly from the original JAR file.

2. APPROACH
In this section, the detail design and implementation information are presented. Specifically various design decisions are summarized at the end of this section.

2.1 Design

The actual analysis phase starts by taking the jar files and reads all the files it contains using unzip java class . So the library extracting the jar file class files only by checking the file extension. The program extracts the targeted files and put it in a temporary directory on the user home folder and it continues by reading the contents of each class file that is already extracted on the previous step. A specific method is used to take a class file and scan all of its method reading every method operations, converts this operation to an ASCII code, and appends that code into a string. After the program finishes scanning all the classes the string will be saved in a text files in the temporary directory.

After a JAR file is converted to text based ASCII symbol string, we need a technique to process it to detect similarities. As we mentioned before, we adopt k-gram technique.

The basic idea of k-gram is: divide a document into *k*-grams, contiguous substring of length *k*, where *k* is a parameter chosen by the user. Then hash each *k*-gram and select some subset of these hashes to be the document's *fingerprints* [1,2]. The purpose of hashing is to convert the k-grams into numbers, which can be comparable easily.

The hash function should be chosen properly so that the probability of collisions is very small. Therefore whenever two documents share one or more fingerprints, it is extremely likely that they share a *k*-gram. Hashing process takes computing power, so researchers try techniques to save computing power.

Since there is a *k*-gram for every ASCII symbol in opcode string, a naive scheme that selected all hashed

k-grams would create an index much larger than the original documents. To select a sub-set of all hashes as fingerprints, Manber's [3] solution is 0 mod p shown above. It does reduce the size of hashes, but the maximum gap between two fingerprints is unbounded and any matches inside a gap are not detected. Heintze [4] proposed choosing the n smallest hashes of all k-grams of a document as the fingerprint. The price for a fixed-size fingerprint set is that only near-copies of entire documents can be detected. To reduce the finger print size generated from opcode string, Winnowing algorithm is chosen [1]. This algorithm has two configuration parameters: k is the k-gram size and w is the window size. The Winnowing starts with k-gram technique, and it selects the smallest hash value from each overlapping window of w sequential hashes. The intuition behind choosing the minimum hash is that the minimum hash in one window is very likely to remain the minimum hash in adjacent windows, since the odds are that the minimum of *w* random numbers is smaller than one additional random number. Thus, many overlapping windows select the same hash, and the number of fingerprints selected is far smaller than the number of windows while still maintaining the guarantee [1]. The commercially available plagiarism detection system, MOSS[5], uses the winnowing algorithm, so the algorithm has been proved scalable.
We adopt the algorithm given in [1], and we choose the window size of four.

2.2 Decisions Made

At the beginning of the project, we were given two major constrains: 1) The inputs to the program will be JAR files; 2) The implements have to implement the specified Java Interfaces [6]. Based on these constrains, some design and implementation decisions are made which relate to source code, size of k-grams, size of winnowing, hashing function.

In order to improve performance, java library methods, for example the hashCode() method, were used wherever possible as these tend to be highly optimized.

3. CONCLUSION

We designed and implemented a Java software similarity detection tool, which can process JAR files to extract .class file, further extract opcode from these .class file.
When software copy happens, variable, constant, class, and method names all could be changed; code sequences can be wrapped in different methods; and methods can be moved to different Java classes, code locations can be moved back and forth. What we are looking for are the essential parts, which can not be easily changed. If these essential parts changes, copying does not exist. The tool catches software copy by analyzing the exact Java bytecode sequences that describe the essential design/algorithm of software. This is the main contribution to software copy detection based on opcode.
Fingerprint is successfully generated from opcode sequence by using k-grams and Winnowing algorithms.
So, Fingerprinting using the winnowing and j-grams methods appear to be an effective way to detect potentially suspicious similarities between jar-files.

4. Future Work

During design and implementation of this software tool, some areas we think should be further explored; and some improvement should be made based on our experiences gained and lessons learned. When analyzing the JAR file, some other components should be assessed. They are .xml file, packaging file, .html file and etc. in JAR file. The finger print generation part of this software tool can be directly used to generate fingerprint for these text based files.

5. REFERENCE

[1]. S. Schleimer, D. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003, pp. 76–85.

[2]. Governments push open-source software, Retrieved Mar21,2010 from http://news.cnet.com/2100-1001-272299.html Michel Ruffin, Christof Ebert, "Using Open Source Software in Product Development: A Primer," *IEEE Software*, vol. 21, no. 1, pp. 82-86, Jan./Feb. 2004.

[3]. Udi Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, San Fransisco, CA, USA, 17–21 1994.

[4]. Nevin Heintze. Scalable document fingerprinting. In *1996 USENIX Workshop on Electronic Commerce*, November 1996.

[5]. The MOSS (Measure Of Software Similarity) System, http://theory.stanford.edu/~aiken/mos

[6]. Elmar Juergens and et al. CloneDetective – A Workbench for Clone Detection Research, Retrieved Mar21,2010 from ,http://www.cs.uoregon.edu/events/icse2009/images/postPosters/CloneDetective.pdf