Anti-plagiarism Project

Eric Scott Freeman

2015-10-21

Contents

1 Introduction

The University of Stavanger uses an application called Autograder, written by Heine Furubotten, to automatically grade students' programming assignments. Autograder works with Github. Whenever a student pushes their code to Github, Autograder will pull a copy of the code and run a series of tests on it. Autograder does not currently have support for plagiarism detection. The goal of this project is to integrate a few anti-plagiarism tools into Autograder, thereby helping the professors save time.

2 Related Work

Unfortunately software plagiarism is a problem both in the classroom and in the workplace. A number of applications have been created to help detect this problem. While these tools can detect similarities in programs, the flagged files must still be manually examined to determine whether or not code was plagiarized.

There are several different general techniques that are used to look for plagiarism. The tools analyzed in this project used either fingerprinting or stylometry.

2.1 Fingerprinting

Several tools use a technique called fingerprinting to detect plagiarism. In fingerprinting algorithms, hashes of n-grams, substrings that are n characters, are saved and compared to help find plagiarism. Not all hashes are stored due to the large number that would be produced.

2.1.1 Winnowing

Moss uses a technique called winnowing to select which hashes to save [1]. In the winnowing algorithm, a window, selection of contiguous hashes, is used to help select which hashes to save. The smallest hash from a window is saved, and then the window moves one hash over. The smallest hash from the next window is often the smallest hash from the previous window. If so it is not saved again. Figure 1 shows an example of how winnowing works. The orange box represents the shifting window. The green box shows whenever a new hash is saved.

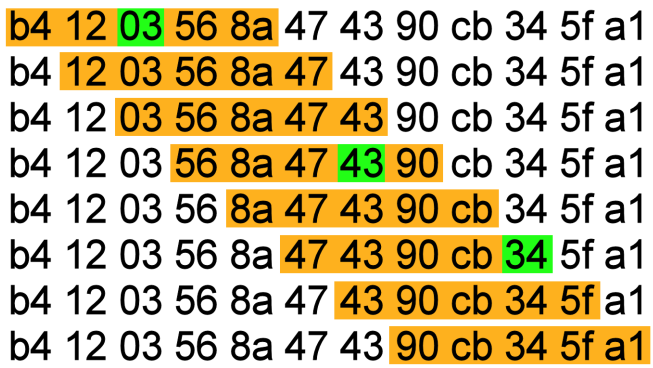

Figure 1: Winnowing.

2.1.2 Running-Karp-Rabin Greedy-String-Tiling

JPlag uses Running-Karp-Rabin Greedy-String-Tiling (RKS-GST) to compare hashes of code in plagiarism detection [2]. RKS-GST was originally used in YAP3, another plagiarism detection tool. In RKS-GST, the Greedy String half of the algorithm forms pairs of substrings, each from a different string. Then the Karp-Rabin half of the algorithm hashes each substring in the pair [3]. This is done to help detect code reordering.

2.2 Stylometry

Another approach is to use code stylometry, which analyzes the style of writing or coding.

2.2.1 Abstract Syntax Trees

Caliskan-Islam, et al. use abstract syntax trees (ASTs) to compare the styles of authors [4]. Things that are easily changed in code, such as variable names, become leaves in the AST, while the structure of the tree is harder to change [4]. Figure 2 shows an abstract syntax tree of the code in Figure 3. Note how the leaves, or circular nodes, in Figure 2 are variable names, constants, and a function name.

Michal Bohuslvek's dupl application uses ASTs to find similarities in code [5]. It looks for any copies of code, not just plagiarism. So if a piece of code is duplicated even in the same file, it will test positive.

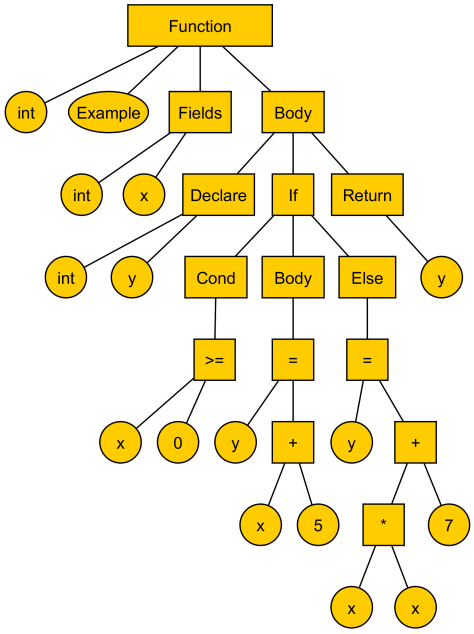

Figure 2: Abstract syntax tree.

```
int Example(int x) {
  int y;

  if (x >= 0) {
    y = x + 5;
  }
  else {
    y = x * x + 7;
  }

  return y;
}
```

Figure 3: Example code.

2.3 Supported Languages

Fingerprinting and string comparison techniques can be used to analyze source code written in languages other than their officially supported languages. Moss can analyze Go code, even though it is not technically supported. Since ASTs need to parse the code, applications which use them are stricter on which languages they can analyze. For example dupl only supports code written in Go. Table 1 shows the languages officially supported by several anti-plagiarism tools.

Tool	Java	Go	C	C++	C#	Python	Perl	others
Moss	✓		✓	✓	✓	✓	✓	✓
JPlag	✓		✓	✓	✓			✓
Plaggie	✓							
SIM	✓		✓					✓
dupl		✓						

Table 1: Officially supported languages by various tools

3 Design

3.1 Getting the tools

Three anti-plagiarism tools were chosen for this project. Moss was the first choice because of the number of languages it supports and its maturity. JPlag and dupl were chosen to give a variety in the algorithms used.

3.1.1 Moss

Moss is a web service, while JPlag and dupl can be run locally. To access Moss, one must send an email to `moss@moss.stanford.edu` containing the following, with `<email address>` replaced with one's actual email address:

```
registeruser
mail <email address>
```

Moss will then send a Moss upload script with a unique user ID, which should be placed in directory `<x>`.

3.1.2 dupl

To download and install dupl, run the following command:

```
go get -u github.com/mibk/dupl
```

dupl requires Go version 1.4 or higher.

3.1.3 JPlag

To download and install JPlag, get the code from `https://github.com/jplag/jplag`. Maven is also required. Download and install it if necessary. Go to the `jplag/jplag/jplag` directory inside the download and run the following command:

```
mvn clean generate-sources assembly:assembly
```

This should create a jar file inside the `./target/` directory called `jplag-x.y.z-SNAPSHOT-jar-with-dependencies.jar` where `x.y.z` is the specific version number of JPlag. Rename this file to `jplag.jar` for use with this project.

3.2 Tool commands

3.2.1 Moss

Here is an example of a Moss command:

```
./moss -l java -m 2 -d ./code/class01/student01/
    assignment01/*.java ./code/class01/student02/
    assignment01/*.java ./code/class01/student03/
    assignment01/*.java > assignment01.txt &
```

The first argument is the Moss upload script. The $-l$ flag signifies that the next argument will be the language the assignments were written in, which in this case is Java. The $-m$ flag signifies that the following argument will be the threshold for Moss, which tells Moss to ignore matches that appear in more than this number of files. In this example, if a piece of code appears in more than 2 files, it is ignored. This is useful if instructors provide some functions or classes for their students to use. The $-d$ option signifies that directories will be compared instead of specific files. In this example, all the java files from three students' assignment 1 will be compared. Moss will

search inside subdirectories. Finally the output from Moss is sent to a text file. The text file will contain a URL that has the results from Moss.

3.2.2 dupl

Here is an example of a dupl command:

```
dupl -t 15 -html ./code/class01/student01/assignment02/
    ./code/class01/student02/assignment02/ ./code/
   class01/student03/assignment02/ > assignment02.html
   &
```

The first argument is a call to dupl. The next argument, $-t$ is dupl's threshold. This is minimum nodes that pieces of code must be before dupl declares them as a duplicates. In this example, it is 15 nodes. $-html$ specifies html output. Next is a list of the directories. dupl will search inside subdirectories. Finally the output from dupl will be sent to an html file.

3.2.3 JPlag

Here is an example of a JPlag command:

```
java -jar ./jplag/jplag.jar -l java17 -t 15 -r ./
   results/lab1 -s lab1 ./students
```

The first three arguments say to run a Java jar file in that location called jplag.jar. $-l$ says which language to use, which in this case is Java 1.7. $-t$ is the minimum number of tokens to match (threshold) argument. The next argument, $-r$ specifies where to save the results. $-s$ says to check all the files in subdirectories with that label. So here any Java files in subdirectories labeled lab1 will be checked. The last argument is the base directory where the code resides.

3.3 Understanding Autograder

To be able to add functionality to Autograder, it was necessary to understand how it works. This meant looking through the code to see what happens when certain events occurred. For example, when the Rebuild button on the teacherresultpage.html is pressed, this calls the `$("#rebuild").click()` function in the teacher.result.page.js file. The click function then posts to the

/event/manualbuild URL. webserver.go handles the post by calling ManualCITriggerHandler() function in ci.go. ManualCITriggerHandler() then starts the daemon, which creates a Docker container and a set of commands to run on the docker container. The docker container pulls the students code and the test code from GitHub and proceeds to run the tests on it.

3.4 Process

Since Autograder was written in Go, this project is also written in Go. Later it was decided to make this project a standalone application that Autograder will call, since another university has expressed interest in using it. The program will call each of the anti-plagiarism tools and store the results. The `golint` application was run against the code for suggestions on making the code follow Go coding conventions. The `go fmt` command was run to clean up the whitespace in the code. A separate Go package will be written for each anti-plagiarism tool. The packages will implement a common interface which will create the commands to send to the tools and will format the results.

The anti-plagiarism application will have two main functions. The first is to call the various anti-plagiarism detection tools, and the second is to check and store the results. It can take an indefinite amount of time for the tools to complete their analysis of the students' code, so it is best for the application to run the commands for the tools as a background process by using the & symbol at the end of each command..

3.4.1 Calling the anti-plagiarism tools

Before calling the anti-plagiarism tools, first the students' code will be pulled from Github to the Autograder server. The directory structure will consist of a base directory containing subdirectories for each class. Inside each class, there will be directories for each student, which will have directories for each assignment. See figure 4. This is similar to how Autograder stores the files in Github, so pulling the code for the anti-plagiarism project will be simplified. Each student's work for an assignment to be compared against the other students' work for that assignment in that class. This will keep the number of files being compared from growing too large.

Next the commands are created for each anti-plagiarism tools based on the location of the code, the language in which the code was written, the

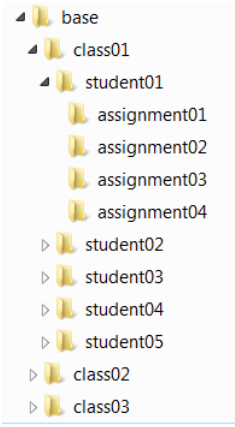

Figure 4: Directory structure.

threshold of the tool, and various other parameters.

3.4.2 Checking for and storing the results

4 Results

5 Issues

A few issues were encountered during the project.

- JPlag will only work with the languages it supports. Therefore JPlag will not accept Go files.
- dupl will not compare files it cannot parse. So files that are submitted with syntax errors could have plagiarized code, but they will remain unnoticed.

6 Analysis

7 Conclusion

References

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

[2] L. Prechelt, G. Malpohl, and M. Philippsen, "Jplag: Finding plagiarisms among a set of programs," *Journal of Universal Computer Science*, vol. 8, 2002.

[3] M. Wise, "Yap3: Improved detection of similarities in computer program and other texts," in *SIGCSE '96 Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, pp. 130–134, 1996.

[4] A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt, "De-anonymizing programmers via code stylometry," in *24th USENIX Security Symposium (USENIX Security 15)*, (Washington, D.C.), pp. 255–270, USENIX Association, 2015.

[5] M. Bohuslvek, "dupl." `https://github.com/mibk/dupl`. Accessed: 2015-09-15.