Abstract

The only fair credit for work is its attribution to the originator. Stealing this credit is in some way equivalent to stealing money or other property. However, this is much easier to perform and much harder to detect. This became particular tempting with the growth of the Internet, since many recent works are readily available for download. Since the Internet is frequently used for plagiarism, it is natural to suggest that it also can be used for plagiarism detection. The present paper shows one way to tackle this problem.

Introduction

The problem of misusing other people's work is omnipresent; from the present work to movies and urban district construction, previous projects are being reused and attributing proper credit is necessary. The extreme form of plagiarism when a project is copied entirely is usually easy to detect; however, parts of projects copied from less well-known sources without proper reference can be very hard to trace.

Background

There is a variety of both text document and code plagiarism detection software available on the market. A popular text plagiarism detector is Plagiarism Detector, available at http://www.plagiarism-detector.com. A popular code plagiarism detector package is CodeScreener, available at http://www.codescreener.com. It works by comparing two packages and determining the code correlation.

However, open-source software dealing with the issue is much less abundant. The software used in the current project is the Incubator-based plagiarism detector. According to the online description [1], "Apache RAT plagiarism detector is command-line tool for searching the code base for possibly plagiarized code using web code search engines".

Unlike the CodeScreener, it only detects direct copying from the Internet; however, it searches the web automatically to determine matches. According to [1], the main goal is to make sure "if you borrow the code from Apache, you are safe and legal." Scanning also improves code reuse, for it is more efficient to reuse entire libraries than separate code fragments.

Approach

The Winnowing algorithm was selected because of its simplicity. As mentioned by Cate Huston [3], the algorithm parses the file into separate *n*-tuples of characters, places into hashes and selects a small fraction of them at random to produce a fingerprint.

For simplicity, fingerprinting is replaced by simpler heuristics, such as the brute force heuristic, where the entire file is scanned, the function heuristic, where the code is parsed into functions or methods, and the comment heuristic, where the comments are ignored. However, the principal part of the Winnowing algorithm, the sliding window, is retained. The window grows until matches no longer can be found. The longest matching string is checked against the criteria for suspects and added to the list if suspicious. Then the sliding window moves to the next element of the heuristic. The process continues until the whole file is parsed.

Design

The code consists of five packages: core, engines, heuristic, report and util. The core package contains the main class, the simplified Winnowing algorithm implementation, and the auxiliary classes to parse the command line and pass the parameters. The engines package consists of classes that feed the code fragments into the Google or Koders search engine, depending on the option. The heuristic package parses the code; the report package generates the report, and the util package contains auxiliary classes for file manipulations. The command line includes the name of the jar file, the codebase path, and optionally the language, the verbose output, the heuristic(s) and the time limit for search.

The following table summarizes the classes and their use.

Package	Class	Description
`org.apache.rat.pd.core`	`PlagiarismDetector`	Main class. Reads options and invokes the heuristic and the engine accordingly.
`org.apache.rat.pd.core`	`PauseListener`	Determines if user requested a pause.
`org.apache.rat.pd.core`	`PdCommandLine`	Parses command line and retrieves arguments. Missing arguments are replaced by default values.
`org.apache.rat.pd.core`	`PlagiarismDetectorReport`	Wrap-up for RAT's IdocumentAnalyzer class
`org.apache.rat.pd.core`	`SourceCodeAnalyser`	Implements the sliding window algorithm
`org.apache.rat.pd.engines`	`ISearchEngine`	Interface implemented by the project's search engine wrap-ups.
`org.apache.rat.pd.engines`	`KodersCodeSearchParser`	Wrap-up for Koders search engine
`org.apache.rat.pd.engines`	`Managable`	Interface for plagiarism criteria implementation
`org.apache.rat.pd.engines`	`RetryManager`	Suppresses network exceptions
`org.apache.rat.pd.engines`	`SearchResult`	Search result
`org.apache.rat.pd.engines.google`	`GoogleCodeSearchParser`	Invokes the search engine and passes code fragment
`org.apache.rat.pd.engines.google`	`MultilineRegexGenerator`	Improved line matching
`org.apache.rat.pd.engines.google`	`RegexGenerator`	Line matching
`org.apache.rat.pd.heuristic`	`BruteForceHeuristicChecker`	The trivial heuristic specifies all the code has to be checked.
`org.apache.rat.pd.heuristic`	`HeuristicCheckerResult`	Specifies which fragments need to be scanned.

org.apache.rat.pd.heuristic	IHeuristicChecker	Heuristic checker interface
org.apache.rat.pd.heuristic.comment	21 language-specific classes	Sets comments to be ignored
org.apache.rat.pd.heuristic.functions	12 language-specific classes	Parses functions to be checked separately
org.apache.rat.pd.heuristic.misspellings	Dictionary	Dictionary loader
org.apache.rat.pd.heuristic.misspellings	MisspellingsHeuristicChecker	Checks comments for misspellings
org.apache.rat.pd.report	HtmlReportGenerator	Generates HTML report
org.apache.rat.pd.report	Report	List of suspicious matches
org.apache.rat.pd.report	ReportEntry	Represents one suspicious match
org.apache.rat.pd.report	TxtReportGenerator	Generates text report
org.apache.rat.pd.report	XmlReportGenerator	Generates XML report
org.apache.rat.pd.util	FileManipulator	Converts code to string for further processing

Results

The program was tested with the simplest Java code; it produced a non-surprising positive result. The following is a chunk of output:

Link: http://www.google.com/codesearch/p?hl=en#p7P5M3b6aAU/skinz/sample_blog_page.html&q=%5E(%5Cs%3F)%2Bpublic(%5Cs%3F)%2Bclass(%5Cs%3F)%2BHelloWorld(%5Cs%3F)%2B%5C%7B(%5Cs%3F)%2B$%20%5E(%5Cs%3F)%2Bpublic(%5Cs%3F)%2Bstatic(%5Cs%3F)%2Bvoid(%5Cs%3F)%2Bmain(%5Cs%3F)%2B%5C((%5Cs%3F)%2BString(%5Cs%3F)%2B%5C%5B(%5Cs%3F)%2B%5C%5D(%5Cs%3F)%2Bargs(%5Cs%3F)%2B%5C)(%5Cs%3F)%2B%5C%7B(%5Cs%3F)%2B$%20%5E(%5Cs%3F)%2BSystem(%5Cs%3F)%2B%5C.(%5Cs%3F)%2Bout(%5Cs%3F)%2B%5C.(%5Cs%3F)%2Bprintln(%5Cs%3F)%2B%5C((%5Cs%3F)%2B%22(%5Cs%3F)%2BHello,(%5Cs%3F)%2BWorld(%5Cs%3F)%2B%22(%5Cs%3F)%2B%5C)(%5Cs%3F)%2B%3B(%5Cs%3F)%2B$%20%5E(%5Cs%3F)%2B%5C%7D(%5Cs%3F)%2B$%20%5E(%5Cs%3F)%2B%5C%7D&sa=N&ct=rx&cd=4

```
            Match 320: public class HelloWorld {
            Match 321: public static void main(String[] args) {
            Match 322: System.out.println("hello, world");
            Match 323: }
            Match 324: }
chunk of code copied:
-----------------------------------------
public class HelloWorld {

 public static void main(String[] args) {
 System.out.println("Hello, World");
 }

 }
```

The program thus works well for checking simple programs for plagiarism; however, it was taking very long time to scan itself. It may not have been sufficiently developed to ensure its suitability for detecting plagiarism of real software.

Conclusion and future work

It would be desirable to make the search more efficient by more thorough parsing of the source code, with emphasis on the structure of the files. It may also be a good idea to include the possibility of renaming variables or functions and slight interchange of operations, since this is very frequently performed when code is plagiarized.

Overall, this is an excellent effort by a Serbian student Marija Šljivovič; the author even mistook it for deployed open-source software.

References

[1] Summer-of-Code project proposals, 2009, http://wiki.apache.org/general/SummerOfCode2009#rat-project
[2] Marija Šljivovič, Apache RAT PD project abstract, http://wiki.apache.org/general/MarijaSljivovic/SoC2009ApacheRatProposal
[3] Cate Huston, Fingerprinting JAR Files Using Winnowing, unpublished paper, http://catehuston.com/work.html