Instructor Copy

Article Review

Zachary Lund

Reference

Ciancarini, P. and Favini, G. P. 2009. Plagiarism detection in game-playing software. In Proceedings of the 4th international Conference on Foundations of Digital Games (Orlando, Florida, April 26 - 30, 2009). FDG '09. ACM, New York, NY, 264-271.

Abstract

Plagiarism is a growing issue in the field of game-playing software. As new ideas and technologies are successfully implemented in free and commercial programs, they will be reused and revisited by later programs until they become standard, but on the other hand the same phenomenon can lead to accusations and claims of plagiarism, especially in competitive scenarios such as computer chess tournaments. Establishing whether a program is a clone or derivative of another can be a difficult and subjective task, left to the judgment of the individual expert and often resulting in a shade of gray rather than black and white verdicts. Tournaments judges and directors have to decide how similar is too similar on a case-by-case basis. This paper presents an objective framework under which similarities between game programs can be judged, using chess as a test case.

Summary

This paper discusses methods for detecting plagiarism in game-playing software. First the authors discuss plagiarism and the importance of detecting it. The dictionary definition of plagiarism is to steal and pass of the ideas or works of another as one's own or to use another's work without crediting the source. Individuals have been accused of plagiarism at some important game playing competitions. This requires that "experts" examine the competing software for similarities and come to a conclusion on whether plagiarism has occurred. The process is manual and normally only happens after someone makes an accusation.

The authors describe the different comparison methods that can be used. If source code is unavailable, the binaries can be decompiled and the generated assembly code can be compared. This allows for some similarities to be discovered, but it is not as good as comparing the original source code. The output of the code can also be compared (black box testing), which is often how suspicious behavior is discovered in the first place.

The authors also discuss what makes a program original. For sufficiently complex games such as chess, the strategies and output for different programs should vary adequately, but for simpler programs like checkers that have an optimal solution, the output from two programs could be very similar without one having plagiarised from the other. This creates a situation that can easily create false positives and false negatives. To date, the detection of plagiarism has always required human intervention.

The authors then introduce several existing systems for detecting plagiarism in code written by students for homework assignments, which they incorporate into their "framework" for detecting plagiarism. These tools use different techniques including fingerprints, string matching, and parse tree matching. Fingerprints simply create a fingerprint for each document based on some metrics, which is not very effective because a small change can create

a large change in fingerprint. String matching is better and compares substrings of the two programs. Parse tree matching takes this further and compares trees that are generated using something like a compiler for the program.

In their framework, they analyze the outputs from the different plagiarism detection tools to build a judgement function which classifies the analysis of severity into three categories of suspicion, confidence, and certainty. They also classify based on the size of the matching code and the location of it in the programs.

To test their idea for a method, they acquire several different open source chess playing agents. They create several modified copies of one of the agents by flipping all of the variable names around and adding additional non-useful code. They start their comparison with a simple trigram-based correlation and then add a tool that does a uniqueness comparison. Next they add a more complex tokenization-based correlation. Their results show that the different open source programs do not share many code similarities. The first two comparisons also do not show much similarity to the copies of the agents that are in fact clones. It isn't until they use the tokenizer comparison that it detects the plagiarism.

Finally, they discuss how these results should be interpreted and propose that better definitions of plagiarism must be established before this process can be automated to a point of not requiring human intervention.

Critique

The authors set out to build an objective framework to judge similarities between game programs. Instead, they use several existing tools and explain how rules are not specific enough remove human intervention altogether. The authors do a great job of explaining why such a tool is important and necessary, and they make their work sound quite appealing, but at the end of the paper, it is clear that it is not that hard to run some existing source code through some existing plagiarism tools and write a few paragraphs about how this is useful and novel.

The authors' abstract is appealing, and their introduction makes a good case for plagiarism detection tools. However, their history of plagiarism in computer games left me diagramming out who accused whom and how the accuser became the accused and then how everyone just forgot about it. It's a nice reminder that much of the problem with plagiarism is political and philosophical and not hard computer science. Unfortunately, this paper is similarly split on its focus.

The authors discuss how plagiarism can be detected using basic tools that have been used to catch cheating students. In my experience correcting college undergrad level homework assignments, those that cheat do not do it in a particularly sophisticated manner. The authors assume that game playing plagiarists will be as good or better as the student plagiarists. I think their assumption is severely lacking for their new audience.

Calling the authors' work a "plagiarism detection framework" seems very generous when all they did was strap a few existing plagiarism detection tools together and interpret the output. For a couple of authors that define plagiarism as using another's work without giving him credit, it's ironic that they don't make it clearer just how little work the authors have done for themselves. To their credit, their framework takes the output from the existing

plagiarism detection programs and performs a judgement on it, something they claim no other system does.

Their results show the similarities between different open source chess playing programs in poorly formatted and difficult to decipher tables, which ultimately reveal the obvious: more sophisticated methods of plagiarism detection produce more accurate results. All of their charts contain numbers that are not labeled and do not provide any sort of scale or upper bound which makes it difficult to determine how good the results are. From what I could decipher, the results for the permutations and obscuration they did on one of the chess playing programs did not produce great results. There was a high correlation for just renaming the variables, but as additional code was inserted, the correlation began to decrease even though the code still did the same thing.

Ultimately, their automated plagiarism detection framework still requires human intervention to make a final decision, which means they only sort of accomplished what they wanted to and said they did. I have to wonder if this approach may be a dead end. However, future work could include generalizations of this approach to work in other plagiarism contexts including general homework coding assignments or even writing. For example, a general description language for specifying specific parameters for different comparisons could be used.