Automatic Generation of Benchmarks for Plagiarism Detection Tools using Grammatical Evolution

Manuel Cebrián, Manuel Alfonseca and Alfonso Ortega*
Universidad Autónoma de Madrid
28049 Madrid, Spain
manuel.cebrian@uam.es, manuel.alfonseca@uam.es, alfonso.ortega@uam.es

Categories and Subject Descriptors: J.1: Education; I.2.2: Program synthesis, Program modification

General Terms: Human factor, Design, Reliability

Keywords: Source Code Plagiarism Detection Tool Assessment, Grammatical Evolution.

1. EXTENDED ABSTRACT

Source code plagiarism detection is a mayor problem in universities worldwide. Although several plagiarism detection tools (PDT) have been deployed, e.g. MOSS [1], JPlag [2] and AC[3], little has been done to assess their quality, because determining the real authorship of a submission corpus is practically impossible for graders.

In this article we present a technique which, fed with a programming assignment, makes use of Grammatical Evolution (GE) techniques [4] to generate a submission corpus made of 40 APL2 functions, each one being a different solution to the proposed assignment. This corpus consists of two subsets. The first, 30 submissions, is 'original', containing solutions to the assignment coded 'from scratch'. The second (14 submissions) is made of plagiarisms, built from one or two solutions taken from the original subset. The whole submission corpus will serve as a benchmark to compare the quality of PDTs.

The 30 original solutions are generated by running a hill-climbing optimization with 30 different random seeds. The operator used by hill-climbing is *mutation with elision*. The optimization consists of finding the shortest program that best fits some mathematical function (the assignment).

We apply three different techniques to generate the plagiarisms. The original solutions are plagiarized using *mutation with elongation* to generate 6 new solutions. This mimics when a single source is changed by adding and replacing a few fragments. The second and third techniques simulate plagiarisms from two sources by means of recombination. The second technique generates 4 new solutions through the *genotypic recombination* of several couples of originals. The third technique mixes other couples by using *phenotypic recombination*. The whole process is depicted in Fig. 1.

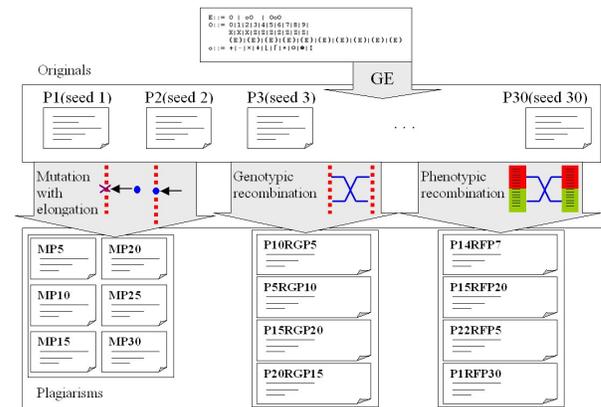We have generated four benchmarks for four different as-

Figure 1: Graphical scheme of the whole process.

signments and fed them to AC, obtaining a remarkable conclusion: AC flagged the overwhelming majority of plagiarisms generated with our algorithm as suspicious, while the original set remained unflagged. This was done by detecting similarities in *trash code*, i.e. erroneous or spurious code. In human detection of plagiarism in real corpora, trash code plays a similar major role: there is a stronger evidence of plagiarism when both submissions share the same trash code than when they share the correct code, because there are many less correct solutions than inaccurate ones. Thus, the probability that two students randomly chose the same inaccurate solution is almost zero.

In conclusion, we have experimental evidence that GE operators are good to simulate the way in which students plagiarize a solution. Design and implementation details, as well as the experimental results can be found in the extended version of this article [5].

2. REFERENCES

[1] A. Aiken et al. Moss: A system for detecting software plagiarism. *University of California–Berkeley*, 2005.
[2] L. Prechelt. et al. Jplag: Finding Plagiarisms among a set of Programs. *Universitat Karlsruhe*, 2000.
[3] M. Freire et al. AC: An Integrated Source Code Plagiarism Detection Environment. arXiv:cs.IT/0703136, 2007.
[4] M. O'Neill et al. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.
[5] M. Cebrian et al. arXiv:cs.NE/0703134, 2007.