A Tool for N-way Analysis of Programming Exercises
C. Andujar    M. Comino    M. Fairen    A. Vinacua
ViRVIG, Universitat Politècnica de Catalunya, Barcelona

Overview
We present a tool to support the grading of programming exercises.

Key ideas:
• Compute syntactic, semantic and functional similarities.
• Embed submissions in 2D, mapping similar code to nearby locations.

Users can:
• Identify clusters (similar submissions).
• Inspect individual submissions.
• Do pair-wise comparisons.
• Do abridged N-way comparisons.
• Sort and grade submissions by similarity.

Exercise example
Exercise 1 [iniCamera] Write a C++ method that initializes a camera
whose frustum encloses the scene while maximizing the viewport occupancy.

void MyGLWidget::iniCamera();

Submission examples
Student i
Student j

Operational similarity
• Based on pass/fail results when running a test set.
• Useful to group submissions by operational correctness.

Character-level similarity
• Similar to diff, i.e. length of matching blocks over total length.
• Useful to group nearly-identical code.

Semantic similarity
• Based on features extracted through a high-level Python API [AVV20].
• Relevant features are found with a $\chi^2$ test based on test pass/fail ratio.
• Useful to group by efficiency (e.g. nested loops), quality (e.g. wrong coordinate space) or robustness (e.g. float equality comparisons).

User Interface

Benefits
Before grading:
• Instructors were able to spot clusters immediately.
• The tool helped checking whether clusters corresponded to uncompleted exercises, similar approaches, or just copies.
• This analysis provided insights to define grading criteria.

During grading:
• Submissions could be graded in the TSP rank order, with similar submissions being graded together. Some submissions could be graded in seconds.
• Instructors reported more consistent scores.

After grading:
• The tool facilitated collecting evidences for plagiarism suspicions.

Limitations
• Our current prototype only supports C++ / GLSL code.
• Useful for exercises requiring small pieces of code (up to 100 lines).
• Dissimilarity matrices have quadratic cost. For massive groups, the approach should operate hierarchically, or on a representative subset.

Future work
• User study to evaluate and quantify these advantages.
• Add further software metrics.
• Add output scores from plagiarism detection software.

Acknowledgments
This work has been funded by the Spanish Ministry of Economy and Competitiveness and FEDER Grant TIN2017-88515-C2-1-R.