Graph based Hybrid Assessment System for Programming Assignments

Soundous Zougari
LTI Laboratory,
National School of Applied Sciences Tangier, Morocco

Mariam Tanana
LTI Laboratory,
National School of Applied Sciences Tangier, Morocco

Abdelouahid Lyhyaoui
LTI Laboratory,
National School of Applied Sciences Tangier, Morocco

ABSTRACT
Programming is a practical process; students need to write a lot of programs in order to master it. However, with large number of students, the assessment of programming exercises leads to extensive workload for teachers making it difficult for instructors to provide constructive and corrective feedback or even additional help when the students need it. In this work, we address the issue of automatic assessment for programming assignments. The goal of which is to provide immediate grading and comprehensible feedback to the learners, while taking some of the workload burden off the teachers. This paper proposes a system combining results from dynamic and static analysis to ensure a reliable and objective evaluation job. While dynamic analysis is based on unit testing framework, the static analysis will quantify the structural similarity between students' programs and the solutions provided by the teacher. In order to perform such comparison, a suitable program representation and an adequate similarity measure will be presented.

General Terms
Graph based assessment, CFG graph representation, graph similarity, grading system.

Keywords
Programming assessment, dynamic analysis, static analysis, CFG similarity measure, automated grading.

1. INTRODUCTION
Computer science is a discipline that is being taught to an increasingly broad audience. This audience is extremely disparate and manifests various needs: high schools students, university students but also professionals continuing education and training of computer science. According to recent studies, learning to program brings enormous benefits for life [1],[2]. Besides improving one's problem-solving abilities, it help acquiring useful traits like perseverance, precision, focus, … and last but not least, it transform us from technology passive consumers into active producers which is incredibly empowering.

As a matter of fact, no student can become a programmer overnight because such learning requires proper guidance as well as consistent practice with the programming exercises. The comments and feedback from teachers about the mistakes they made are crucial to acquire adequate skills in programming and enhance their knowledge.

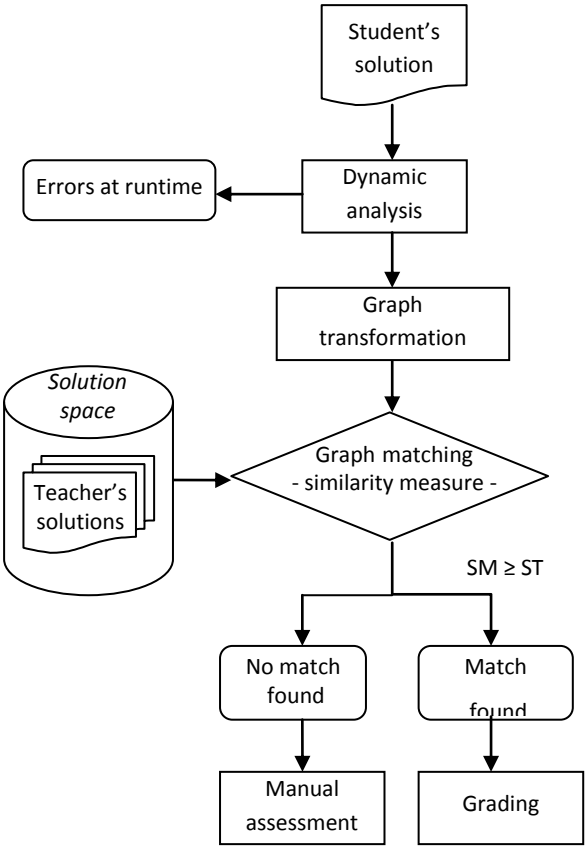
However, due to the large number of students enrolled in such courses, instructors find themselves rapidly overloaded. Indeed, manually tracking errors for every student's program is difficult and time-consuming. As a consequence, the delay between the time of submitting the student code for a problem and its feedback is also increased. Moreover, manual assessment of student coding is prone to errors or omissions due to the fatigue and the repetitive nature of the task [3].

Automatic assessment systems are also of particular interest in the context of e-learning [4]. Indeed, when the learning process is mediated by a web-based learning system, the delay on providing feedback may lead to student's frustration or course abandonment. Therefore, fast and reliable automated assessments are particularly desirable. To address these issues, researchers have been focusing on automating the process of assessing learners' productions. The first reference comes from Hollingsworth who published on the subject in 1960 [5]. The idea spread quickly and numerous assessment systems have been developed [6][7][8]. Unfortunately, these systems are neither generic nor configurable and most of them are not available to the general public, which is why we seek to develop an assessment system.

In this context, we propose a reliable and objective method of assessing learners' productions that not only will reduce the workload for teachers but also provide instant grades and useful feedbacks to students throughout their learning process. Concerning the practical domain, it was opted for introductory programming courses for several reasons. Besides the fact that these courses are the core of any engineer's training, this is a domain where assessment is of a great complexity, mainly because it is characterized by the multitude of solutions to a given problem.

The remainder of the paper is organized as follows: section 2 describes the proposed hybrid approach merging results from two program analysis methods. Afterwards, the 3rd section proposes a suitable program representation whereas the 4th section addresses the programs similarity issue. Finally, Section 5 gives conclusions and discusses about the future research.

2. PROPOSED ASSESSMENT METHOD
The validation of computer programs is a crucial part in the cycle of their development. Two verification and validation techniques have stood out in recent years: dynamic analysis and static analysis. The main difference between these two approaches is that the dynamic analysis requires the execution of the program to check its accuracy, unlike the static analysis that examines a program without executing it. In a previous work, it was deduced that the strengths and weaknesses of the dynamic and static approaches are complementary [9]. Therefore, an original combination of these two techniques was proposed. In this combination, the dynamic analysis reports errors at runtime, whereas the static analysis evaluates the structural properties of the programs. Figure 1 resumes the assessment approach. The student' proposed solution go through all the process even if it generates errors from the start.



SM : Similarity measure
ST : Similarity threshold

Fig 1: Proposed assessment approach.

To perform dynamic analysis, students' programs are run through a set of data, and afterwards their outputs are compared to the predefined answers. The approach is described in more detail in a previous paper where it was suggested the use of xUnit, a dedicated framework to automate and conduct tests in a given language [9].

On the other hand, to evaluate the structural properties of the programs (static analysis), the similarity degree is measured by comparing the assessed program to programs belonging to the solution space provided by the teacher or expert. A solution space is a set of paths representing the different possible approaches for the same exercise. It can contain the correct solutions as well as the incorrect ones. It is made by an expert and has deemed pedagogically interesting approaches. If a match is found; similarity measure is superior to a threshold defined by the teacher, the student's program will be graded automatically, or else the program is submitted to the teacher for manual assessment. In the last case, the students' solution can be added to the solution space if it is judged pedagogically interesting or a recurrent incorrect solution. This approach will gradually decrease human intervention.

This method requires two steps; the passage through the graphical representation of the compared programs, which is addressed in section 3 and a similarity or a matching process. More details are given in section 4.

3. PROGRAM'S GRAPHICAL REPRESENTATION

Since the intention is to assess students' productions in introductory programming courses, the programs are represented with control flow graphs. The Control Flow Graph (CFG) is a directed graph where each node represents a basic block i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block. Directed edges are used to represent jumps in the control flow. It highlights loops, conditional statements and branches. A path in this graph represents a program implementation scenario.

The program illustrated in Figure 2, is used to provide an example for control flow graphs. This is a simple program that initializes two variables x and y, and executes 2 commands repeatedly in the while loop until y is greater than or equal to 10.

```
voidmain() {
  int x = 0;
  int y = 1;
while (y < 10) {
  y = y * 2;
  x = x + 1;
    }
printf("%d",x);
   printf("%d",y);}
```

Fig 2: Program example.

It is noteworthy that in the flow control approach, the focus is on the sequencing of operations in a process. Control flow graphs are used as models to describe the structure of computer programs. They are used both for static analysis [10] and as a model for program coverage. Therefore, it's a suitable representation for structural comparison. However, other types of graphs will not be definitively excluded because they can be interesting for future modifications in the proposed system.

The program corresponding CFG is displayed in Figure 3.

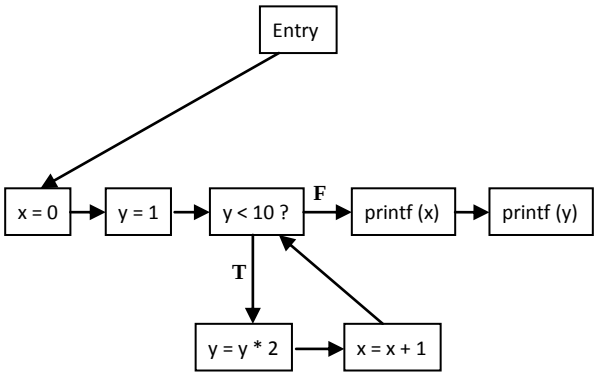


Fig 3: Control Flow Graph.

This research call for a quantitative measure of the 'similarity' of two programs; student proposed solution and teacher's solutions.

Once the Control Flow Graphs are extracted from the programs, the similarity measurement becomes a graph problem.

First, let's briefly introduce some notion of the graphs, which will recur throughout the next section.

4. PROGRAM'S GRAPH SIMILARITY AND GRADING

4.1 Graph Similarity Measure

Graph similarity has numerous applications in diverse fields (such as social networks [11], image processing [12], biological networks [13], chemical compounds, computer vision,…), and therefore there is a number of proposed algorithms and measures devoted to the graph similarity problem. The proposed techniques can be classified into three main categories: edit distance/graph isomorphism, feature extraction, and iterative methods. A short overview of similarity measures for graphs can be found in [14][15][16].

Choice of a similarity measure to be used in some context is often guided by its usefulness in practice. Since there is a need of a graph similarity method that uses as parameters the labels of the nodes, the direction of the edges and the number of the common edges, it was opted for a specific similarity measure for graph nodes called neighbor matching. This method has properties relevant for our needs that other measures lack [17].

In this section will be outlined a graph similarity measure that uses the structural similarity of local neighborhoods to derive pairwise similarity scores for the nodes of two different graphs. More precise definition will be given ahead, but first, a brief definition for some notion of the graphs, which will be needed throughout this section.

4.1.1 Definitions

1. A **directed graph** G = *(V, E)* is defined by its set of nodes *V* and its set of edges *E*. There is an edge between two nodes *i* and *j* if (*i, j*) ∈ *E*. The node *i* is an in-neighbor of node *j* and the node *j* is an out-neighbor of the node *i* if (*i, j*) ∈ *E*. An in-degree *id(i)* of the node *i* is the number of in-neighbors of *i*, and an out-degree *od(i)* of the node *i* is the number of out-neighbors of *i*.

2. The **similarity measure** s is a function *s: D1 × D2 → R* where D1 and D2 are possibly equal sets of objects and R being a real number between 0 and 1 that captures intuition well; a higher value of similarity measure should imply a higher similarity in some intuitive sense.

3. Similarity measure over the nodes of two graphs can be represented by a **similarity matrix** $X = [x_{ij}]$ of dimension $|V_A| \times |V_B|$ with the element x_{ij} denoting a similarity of the nodes $i \in V_A$ and $j \in V_B$.

4. Let A and B be two finite sets of arbitrary elements. A **matching of elements of sets A and B** is a set of pairs $M = \{(i, j) | i \in A, j \in B\}$ such that no element of one set is paired with more than one element of the other set. For the matching *M* we define enumeration functions $f : \{1, 2, \ldots k\} \rightarrow A$ and $g : \{1, 2, \ldots k\} \rightarrow B$ such that $M = \{(f(l), g(l)) | l = 1, 2, \ldots, k\}$ where $k = |M|$.

4.1.2 Similarity graph algorithm

The algorithm that is proposed in this paper is based on *N.Mladen* [17] research paper. It derives from the neighbor matching technique. The mentioned algorithm relies on the simple following concept: two nodes $i \in G_A$ and $j \in G_B$ are considered to be similar if neighbor nodes of *i* can be matched to similar neighbor nodes of *j*, hence the name neighbor matching.

First, it iteratively measures the similarity of nodes in the students' and teacher' graphs and finally calculates one similarity score using that similarity measures. Followings are the equations that are used to calculate the similarity of nodes and we will explain what will happen in each one:

$$x_{ij}^{k+1} \leftarrow \frac{s_{in}^{k+1}(i,j) + s_{out}^{k+1}(i,j)}{2} \quad (1)$$

Where

$$s_{in}^{k+1}(i,j) \leftarrow \frac{1}{m_{in}} \sum_{l=1}^{n_{in}} x_{f_{ij}^{in}(l) g_{ij}^{in}(l)}^{k}$$
$$s_{out}^{k+1}(i,j) \leftarrow \frac{1}{m_{out}} \sum_{l=1}^{n_{out}} x_{f_{ij}^{out}(l) g_{ij}^{out}(l)}^{k} \quad (2)$$

And

$$m_{out} = \max\big(od(i), od(j)\big) \quad m_{in} = \max\big(id(i), id(j)\big)$$
$$n_{out} = \min\big(od(i), od(j)\big) \quad n_{in} = \min\big(id(i), id(j)\big)$$

The first equation (1) will calculate the similarity of i^{th} node of graph G_A and j^{th} node of graph G_B in *(k+1)* iterations. As might be seen, we need to calculate $s(i,j)_{in}$ and $s(i,j)_{out}$ in *(k+1)* iterations first. $s(i,j)_{in}$ is the in degree similarity of node *i* in G_A and *j* in G_B. $s(i,j)_{out}$ is the out degree similarity of node *i* in G_A and *j* in G_B. f_{ij}^{in} and g_{ij}^{in} are the enumeration functions of the optimal matching of in-neighbors for nodes *i* and *j*, and analogously for f_{ij}^{out} and g_{ij}^{out}.

In the equations (2), in the case when:

$$m_{in} = n_{in} = 0$$

or $m_{out} = n_{out} = 0$

We have:

$$\frac{0}{0} = 1$$

The initial similarity values x_{ij}^{0} are set to 1 for each *i* and *j*.
After initializing the in-degree and out-degree similarity matrices, the similarity matrix is initialized using those two matrices.
Afterwards comes the iteration and calculation of each node similarity until the similarity scores converge. To check that, a value called epsilon (a chosen precision) is used and the following termination condition:

$$max_{ij} \left| x_{ij}^{k} - x_{ij}^{k-1} \right| < \varepsilon$$

The similarity matrix $[x_{ij}]$ reflects the similarities of two graphs G_A and G_B nodes.

The similarity of the graphs can be defined as the weight of the optimal matching of nodes from G_A and G_B divided by the number of matched nodes [17].

In order to give student's a grade, a feedback that not only students easily understand but also is mandatory for the achievement of the course, we will use the obtained similarity information in the automated grading.

The similarity value, a number between 0 and 1, can be considered as an intuitive feedback [18]. In fact, the feedback could be that:

- The solution is dissimilar (**0-0.5**),
- The solution is roughly similar (**0.5-0.7**),
- The solution is similar (**0.7-0.9**),
- The solution is very similar or corresponds to one of the teachers' solutions (**0.9-1**).

4.2 Grading

The most common problem when managing large numbers of students in programming classes is the grading. Grading environments include document flow, grading itself and record keeping. Document flow includes distribution of assignments, programs submission, and the return of graded programs with feedback. The assessment system presented in this paper will automate the grading process with little to no human intervention.

Generally, the grades are expressed at a scale from 0 to 20. There may be different grading settings depending on aims of the exercise and goals of teachers. As mentioned above, the focus is on introductory programming courses, therefore the teacher expect from students to write working programs and use the aspects covered in the course and/or the requirements specified in the exercise. In fact, a program producing a right output may not meet the programming specification. For example, the students are required to implement a program that outputs ten characters '*' by using an iteration structure. However, some students use ten output statements instead of an iteration structure.

As mentioned earlier in this paper, even if a student fails to provide a working program that gives correct results for given test cases (static analysis), his solution will be further examined through dynamic analysis. However, the student will be penalized for the problems that prevent the program from compiling, running, or passing a test case.

In this case, two penalty parameters were added to the grading model P_1 and P_2.

The first grading penalty P_1 is used when the teacher wants to evaluate if a program is working (compiling, running or test cases). Whereas the second penalty parameter calculates how close is a solution to the teacher solution.

The proposed grade is a linear combination of different scores measured for the student's solution, which provide an equation of the following form:

$$\boldsymbol{G} = P_1 * x1 + P_2 * x2 \qquad (3)$$

Where

G is the automated grade,

x1 is the weighted sum of the automated testing cases passed. It is expressed in the interval [0, 1],

- *x2* is the maximal value of similarity between the student's solution and the teacher solutions, also in the [0,1] range.

It should be noted that different choices for the coefficients P_1 and P_2 could be proposed as long as $\boldsymbol{P_1+P_2=20}$. However, it is preferable to let the teacher tune the coefficients P_1 and P_2 so that the behavior of the predictive model corresponds to the teacher's grading style and the exercise goals.

4.3 Feedback

Immediate and corrective feedback is vital in the learning process. It is especially important for novice programmers to not only know whether their programs are correct, but also the details about the errors, to help point them in the right direction. Based on the feedback, they can become aware of their difficulties and what they need to further study and improve [19].

Through the dynamic testing, the information is gathered then displayed for the student once the assessment is completed. It points out whether a student program passed the dynamic testing or not. If a test fails, detailed information on that test is included. This information includes a copy of the input supplied to the program and the correct output the program should have generated. Once the students see the input data that resulted in errors, they have the opportunity to learn something about the nature of good testing data.

Although it might seem that the instructor invests more time writing a testable assignment specification and developing the grading program, these costs are expected to be amortized over multiple courses and assignments. Moreover, the assessment provides the teacher with a feedback channel that shows how learning goals are being met. It also ensures for an outside observer that students achieve those learning goals.

5. CONCLUSION AND FUTURE WORK

The above presented method merges results from dynamic and static analysis to ensure a reliable and objective evaluation job. In one hand, the dynamic analysis is carried out using unit testing framework making the process flexible and reusable. On the other hand, the static analysis focus on finding structural similarities between students' and teachers programs after transforming them into control flow graphs.

The proposed assessment system has been developed and is at the current moment undergoing some encouraging testing with real students' exercises to assess its usability and integration to an automated submission system. This experiment will also allow us to evaluate its weaknesses and therefore improve it.

Next, we would like to focus on quantifying the advantages from using such assessment system in the introductory programming course. A quantitative and qualitative analysis of students' performance and the robustness of the assessment mechanism will provide further insight into the proposed system.

6. REFERENCES

[1] Scherer, R., Siddiq, F., & Sánchez Viveros, B, The cognitive benefits of learning computer programming: A meta-analysis of transfer effects, Journal of Educational Psychology. Advance online publication,2018,

[2] K. Heggart, Coded for success: the benefits of programming among school students, June 2014,

[3] Higgins, S., Hall, E., Baumfield, V., Moseley, , A meta-analysis of the impact of the implementation of thinking skills approaches on pupils, Research Evidence in Education Library, 2005,

[4] Allen. I.E, Seaman. J, Class Differences: Online Education in the United States, Newburyport, MA: Babson Survey Research Group and The Sloan Consortium Green, 2010,

[5] J. Hollingsworth. Automatic graders for programming classes, Communications of the ACM, 3:528–529, October 1960,

[6] C.Douce, D.Livingstone, and J. Orwell, Automatic test-based assessment of programming: A review, Journal on Educational Resources in Computing (JERIC), 2005,

[7] Rohaida Romli, Shahida Sulaimanb, Kamal Zuhairi Zamlic, Improving Automated Programming Assessments: User Experience Evaluation Using FaSt-generator, Procedia Procedia Computer Science 72, 186 – 193, 2015,

[8] David Insa, Josep Silva. J.A, Automatic assessment of Java code, Computer Languages, Systems & Structures Vol 53, Pages 59-72, 2018,

[9] Soundous Zougari, Mariam Tanana, Abdelouahid Lyhyaoui, Towards an automatic assessment system in introductory programming courses, 2016,

[10] S. Rao Kosaraju, Analysis of structured programs. J. Comput. Syst. Sci.,9(3) :232-255, 1974,

[11] Sadia Tariq, Muhammad Saleem and Muhammad Shahbaz, User Similarity Determination in Social Networks, 2019.

[12] Jieqi Kang, Image processing and understanding based on graph similarity testing: algorithm design and software development, 2017.

[13] Mohammad Shafkat Amin, Russell L. Finley, Hasan M. Jamil, Top-k Similar Graph Matching Using TraM in Biological Networks, 2012,

[14] D. Conte, P. Foggia, C. Sansone and M. Vento , Thirty Years Of Graph Matching In Pattern Recognition, International Journal of Pattern Recognition and Artificial Intelligence Vol. 18, No. 03, pp. 265-298, 2004,

[15] Vincenzo CARLETTI, Exact and Inexact Methods for Graph Similarity in Structural Pattern Recognition, 2016,

[16] Danai Koutra et al., Algorithms for Graph Similarity and Subgraph Matching, 2011,

[17] M. Nikolic, Measuring Similarity of Graph Nodes by Neighbor Matching, Journal Intelligent Data Analysis, Vol 16 Issue 6 Pages 865-878, 2012,

[18] MilenaVujošević-Janičića, Mladen Nikolić, Dušan Tošić, Viktor Kuncakb, Software verification and graph similarity for automated evaluation of students' assignments, Information and Software Technology Vol 55, 2013.

[19] Tiantian Wang, Xiaohong Su, Peijun Ma, Yuying Wang, Kuanquan Wang, Ability-training-oriented automated assessment in introductory programming course, Computers and Education vol.56 issue 1 pages 220-226, 2011.