International Journal of Electrical and Computer Engineering (IJECE)
Vol. 12, No. 3, June 2022, pp. 2867~2875
ISSN: 2088-8708, DOI: 10.11591/ijece.v12i3.pp2867-2875

Validity of a graph-based automatic assessment system for programming assignments: human versus automatic grading

Soundous Zougari, Mariam Tanana, Abdelouahid Lyhyaoui
LTI Laboratory, National School of Applied Sciences of Tangier, Abdelmalek Essaadi University, Tetouan, Morocco

Article Info

Article history:

Received Dec 30, 2020
Revised Jan 3, 2022
Accepted Jan 23, 2022

Keywords:

Automated grading
Automatic assessment
Computer aided education
Graph-based assessment system
Programming assignments

ABSTRACT

Programming is a very complex and challenging subject to teach and learn. A strategy guaranteed to deliver proven results has been intensive and continual training. However, this strategy holds an extra workload for the teachers with huge numbers of programming assignments to evaluate in a fair and timely manner. Furthermore, under the current coronavirus (COVID-19) distance teaching circumstances, regular assessment is a fundamental feedback mechanism. It ensures that students engage in learning as well as determines the extent to which they reached the expected learning goals, in this new learning reality. In sum, automating the assessment process will be particularly appreciated by the instructors and highly beneficial to the students. The purpose of this paper is to investigate the feasibility of automatic assessment in the context of computer programming courses. Thus, a prototype based on merging static and dynamic analysis was developed. Empirical evaluation of the proposed grading tool within an introductory C-language course has been presented and compared to manually assigned marks. The outcomes of the comparative analysis have shown the reliability of the proposed automatic assessment prototype.

Corresponding Author:

Soundous Zougari
Innovative Technology Laboratory, National School of Applied Sciences of Tangier, Abdelmalek Essaadi University
Route de Ziaten Km 10, Tanger Principale, BP: 1818 – Tanger, Tetouan 93000, Morocco
Email: soundous.zougari@gmail.com

1. INTRODUCTION

Stephen hawking once said "*whether you want to uncover the secrets of the universe, or you want to pursue a career in the 21st century, basic computer programming is an essential skill to learn*". Indeed, in an era marked by rapid advances in our technologies, manipulating those technologies has emerged as a key skill integral to many jobs in the present time and in the near future. However, learning programming is challenging: high failure and dropout rates are common in those courses. A strategy guaranteed to deliver proven results, for students as well as teachers, has been intensive and continuous practice on solving programming exercises [1], [2]. For students it helps them to understand their learning progression and difficulties. Whereas, for teachers, it enables them to see students' learning progressions and eventually to give personalized support and adjust the teaching materials accordingly.

This is a strategy that holds an extra workload for the teachers with huge numbers of programming assignments to evaluate and provide with feedback in a timely manner. Apart from being time-consuming for the teacher, manual assessment hinders the consistency and accuracy of assessment results as well as it allows "unintended biases and a diverse standard of marking schemes" [3]. All of these issues called for the automation of the assessment process.

Furthermore, in recent years, massive open online courses (MOOCs) have become a significant force within higher education. They have quickly gained popularity, expanded, and evolved as they bear a tremendous potential for teaching programming to a large and diverse audience [4]–[6]. A potential put to the test in this difficult time with the coronavirus (COVID-19) pandemic [7]. We are witnessing the world's largest distance learning experiment, since schools and universities worldwide were forced to shift rapidly from presential to online learning in the midst of this global health crisis. Unfortunately, for programming assignments, MOOCs remain limited in their ability to evaluate and give feedback. In fact, the learning platform would be reduced to merely offering optional ungraded exercises if there was no method for automatic assessment of programming assignments [4], [8].

In the light of this, numerous automatic assessment tools have been developed by various researchers and academicians. However, these systems are neither generic nor configurable and most of them are not available to the general public, which is why we have developed our own assessment system. Although the automatic approach theoretically overcomes all the drawbacks of the manual approach, the quality of the provided assessment requires some investigation.

The remainder of this paper is structured as follows: section 2 explores some of the work that has been done in the automated programming assessment area. Section 3 describes the proposed system approach before putting the prototype system to the test and analyzing the results in detail in section 4. Last, section 5 ends with some reflections on the produced research and suggestions for further work.

2. REVIEW OF RELATED WORK

Several approaches to tackle automatic programming assessment (APA) are reported in the literature, such as journals, conference articles and online resources. A systematic literature review can be useful in providing a wide view of the existing tools, the identification of research needs and guiding the development of new tools. Our focus is solely on the automatic assessment tools applied to programming assignments.

This topic has been of interest to computer science educators from 1960s [9] and has continued to gain vast attention till present [10]-[23]. Indeed, a variety of systems have been developed to address the problem of automatic and semi-automatic evaluation of programming assignments. To see an evolution, it is necessary to take a temporal perspective.

Douce *et al.* [10] provide a systematic review of the literature on the automatic programming assessment systems developed from their inception up to 2005 and categorize them according to age into three broad generations. In each of the three identified generations, these systems adopted more advanced technologies correlating with cutting-edge technologies used for program development in each time frame:

- The first-generation systems represent the earliest attempts to automate testing and were considered genuine advancements at the time [10]. The assessment was only made considering a right or a wrong answer. However, their usability was restricted to their specific computing research facilities.
- The second-generation systems are characterized by command-line-based tools, sometimes used in association with locally built and maintained Graphical user interfaces (GUI).
- The third-generation systems utilize web-based technologies, and sometimes provided additional support for educators in the form of assessment management and reporting facilities.

Ihantola *et al.* [12] complemented this review with APA systems developed during the 2006 to 2010 period. They divided APAs in two main categories: automatic assessment systems for programming contests and automatic assessment systems for computer programming education [13]. Moreover, the authors systematically collected and grouped the features and improvements in APA systems from the selected review period. Some of the reported features are programming languages, learning management systems, defining tests and resubmissions.

Over the last few years, and with the proliferation of new assessment tools, studies have put their focus on identifying and classifying them. This has eased the process when reviewing approaches and features provided by each tool. For instance, Souza *et al.* [14] have proposed three classification schemes for the reviewed tools: by assessment type, by approach, and by specialty. Besides, they have identified the main characteristics of the assessment tools such as:

- Main Features: electronic submission, automated checking, instant feedback, automated marking,
- Types of verification: dynamic verifications and static verifications.
- Interfaces: command line interface (CLI), graphical user interface (GUI), web user interface (WUI).
- Supported programming languages.

Although we believe a lot has been done since 2016, recent surveys from 2018 by Lajis *et al.* [18] and by Ullah *et al.* [19] provide little new to the work of Souza *et al.* [14]. The former classified several automated assessment systems under the same main approaches, while the latter discussed the strengths and limitations of 17 automated assessment tools. Among the most popular reviewed tools we find, AutoLEP [22], Web-CAT [23], BOSS [24], Quimera [25], DOMjudge [26], and Automata [27].

This research has helped us to understand the current state-of-the-art in assessment tools for programming assignments, and also to identify important features of already built tools and some future directions. First, a foregone conclusion is the plethora of existing automatic programming assessment systems and still growing. However, these tools are rarely used beyond the institutions in which they were created. The main reason for this is related to availability and adaptation issues. Only few systems are open-source, or even freely available. Other prototypes were designed in the context of a doctoral research but were no further developed. Another obstacle for an APA tool broad adoption is that instructors have difficulty integrating systems built for specific requirements, especially when the notation and methodology of such systems does not precisely fit into their courses. Developing a new assessment tool for each programming language and instructor's assessment objectives is not plausible. We, therefore, suggest developing a flexible and parameterizable tool that will not only be able to automatically assess a variety of programming languages but also adapt to each instructors need and a course's main target group.

Another interesting conclusion of this work is that in general the APA features can be organized according to whether they need execution of the program; dynamic analysis or can be statically evaluated from the program code; static analysis. Both of these approaches present undeniable advantages [18], [28], but also some major drawbacks. Therefore, our approach falls within a third kind of automatic assessment tool that has been less investigated: hybrid analysis. The general idea is to merge results from the static and dynamic analysis. Thus, the student's program output is not only analyzed but also its source code. This approach overcomes the limitation of static and dynamic approaches. Further information will be given in the next section.

3. THE PROPOSED SYSTEM

Our proposed system combines results from dynamic and static analysis to improve the quality and precision of the automated assessment and overcomes the individual weaknesses of these approaches [29], [30]. The dynamic analysis is carried out using unit testing framework making the process flexible and reusable. In fact, students' programs are run through a predefined set of data, and afterwards their outputs are compared to the expected 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 [31]. XUnit is additionally used to supply significant and comprehensible feedback. Generated feedback follows directly from the detected failed dynamic tests. Information like the line number, the error type, program trace that introduces the error, and values of variables along this trace can help the student refine its proposed solution.

On the other hand, static analysis helps us evaluate the structural properties of the programs. The idea is to measure the similarity degree by comparing the assessed program to programs belonging to the solution space. A solution space is a set of programs, provided by the teacher, representing the different possible solutions for the same exercise. The main problem of this approach is the diversity of the solutions. To tackle this problem, semantic preserving transformations are performed on the student program and the solution space. Through this process we try to eliminate syntactic variations and represent in a uniform way, semantic equivalent programs that use the same algorithm. However, there are cases where a given student proposes a solution that was not foreseen in the solution space. Then, the assessor's intervention is required to analyze it and add it to the solution space if it found pedagogically relevant. These interventions should be rare in the context of introductory programming courses and will decrease in time, once the solution space has reached a satisfactory level of maturity. Figure 1 resumes the assessment approach. The student' proposed solution go through all the process even if it generates errors from the start.

In order to facilitate the program matching, the student program and the instructor program are firstly transformed into an intermediate representation. We opted for control flow graph (CFG) which is a graph-based representation where nodes represent blocks of code and edges represent transfers of control between blocks. In order to measure similarity of the programs, we adopt the concept of merging node content similarity with graph nodes topological similarity. Our similarity is based on a particular graph node similarity measure called neighbor matching. This measure is based on iterative calculation of similarity and the principle that two nodes are as similar as their neighbors are [32]. The calculation of similarity is detailed in a previous work [31] and given by (1).

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

where x_{ij} is the calculation of nodes i and j similarity in (k+1) iterations. y_{ij} is the similarity of nodes i and j contents. $S_{in}^{k+1}(i,j)$and $S_{out}^{k+1}(i,j)$ are respectively, the in-neighbors and the out-neighbors similarity of nodes i in G1 and j in G2 in (k+1) iterations. Also, we set $x_{ij}^{0} = y_{ij}$. Both the similarity of nodes and the similarity of CFGs take values in the [0, 1] interval. More details were given in a previous paper [31].

Regarding the grading step, unlike many other systems, our tool has not been developed for a predefined grading style. We have used the similarity information and introduced two penalty parameters P_1 and P_2. The first grading penalty 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's solution. This practice of grade calculation weighting is an important technique used to make program grading more parameterized and personal to the evaluator [23]. In sum, the grade equation is a linear combination of various scores, calculated for the student's solution in (2):

$$G = P_1.\ x1 + P_2.\ x2 \qquad (2)$$

where G is the automated grade of 0 to 10; $x1$ is a value retrieved at the end of the dynamic analysis and it represents the weighted sum of the automated testing cases passed (It is expressed in the interval [0, 1], "1" meaning all tests were passed successfully); $x2$ is the maximal similarity value between the student's solution and the teacher proposed solutions (also in the [0, 1] range); P_1 and P_2 are the dynamic and the static assessment penalty parameters, respectively. It should be noted that different choices for the coefficients P_1 and P_2 could be proposed as long as P_1+P_2=10. However, we prefer to let the teacher tune the coefficients P_1 and P_2 so that the behavior of the predictive model corresponds to the teacher grading style and the exercise goals.

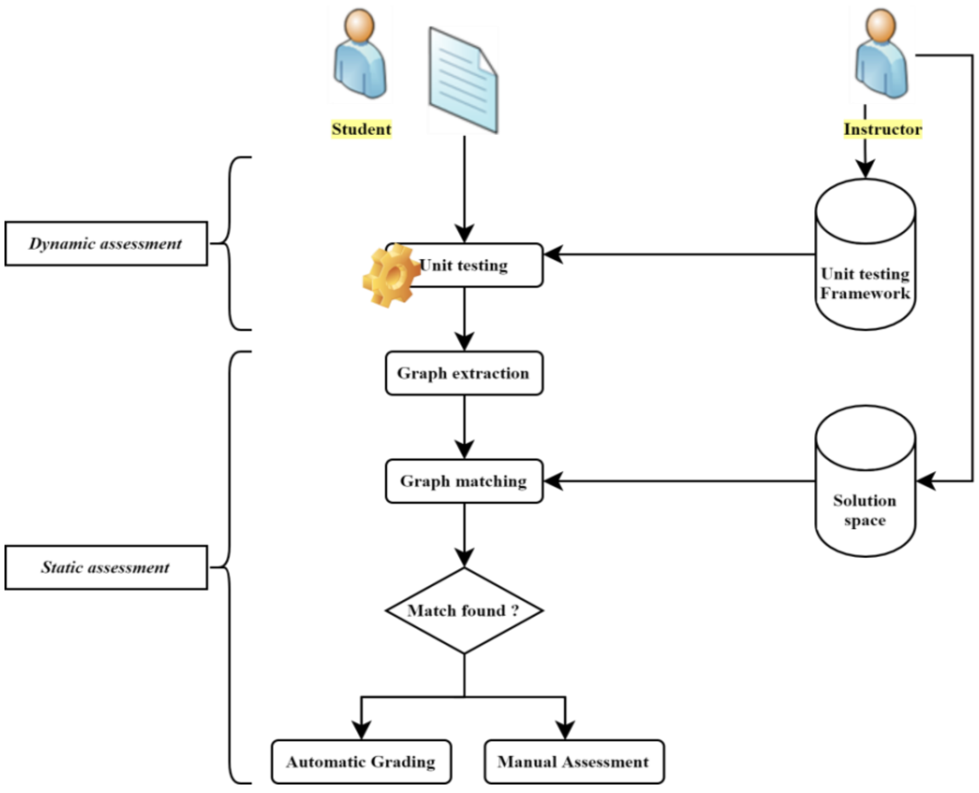

Figure 1. Proposed automatic assessment system

4. RESULTS AND DISCUSSION

To have a real estimation of the proposed system efficiency and reliability, we have carried out an experiment with some exams of real university programming courses. Through this experiment, we collected the teacher's manually graded exercises. Then we marked them automatically with our tool to evaluate the closeness of the proposed automated assessment system to the human rater method.

4.1. Method

The participants were 50 first-year, randomly selected, engineering students of a C programming module in their second year at the National School of Applied Sciences of Tangier, Morocco. The used probability sampling technique is the simplest of all; however, its simplicity is also its strength. The sample is non-biased, and the findings are most generalizable among all probability sampling approaches since the sampling frame is not subdivided or partitioned.

We have chosen the C-language for our tests because it is a flexible and strong general-purpose programming language. It is efficient, portable, and operates on any platform. Moreover, it is still considered in 2020, in the top 5 best programming languages to learn and more importantly, it is a great way to begin a programming career [33]. We have retrieved three exercises of escalating difficulty from the C-language introductory course. Here are their contents:

- Exercise no 1: write a C program that get two integer numbers, multiply both of the integers and display the product.
- Exercise no 2: program that asks the user to enter two strings as operands and then displays a list of the common letters (characters) of the two strings.
- Exercise no 3: program that implements the Ackermann function (function which returns the value of *A(m, n)* using recursion). The Ackermann function is defined as (3).

$$A(m,n) = \begin{Bmatrix} n+1 & if\ m = 0 \\ A(m-1,1) & if\ m > 0\ and\ n = 0 \\ A(m-1, A(m,n-1)) & if\ m > 0\ and\ n > 0 \end{Bmatrix} \quad (3)$$

In order to use our proposed system, we have prepared adequate test-cases for every exercise and provided each exercise with related instructor solutions. Concerning the grading penalty parameters, we preferred to assign the same weight to the 2 penalty parameters. That is to say, both the static and dynamic analysis will have the same impact on the grade. This decision was motivated by the fact that there are divided opinions when awarding points based on a working program or a used knowledge or skill. It is also a good opportunity to measure the proposed system efficiency and objectivity with the minimum required resources for scoring.

4.2. Results

The automatic assessment has worked quite well and we have been able to assess all the students' submissions automatically. Table 1 shows some descriptive statistics of the 50 submissions that were assessed manually as well as automatically, with the purpose of having a broad understanding of both assessments' behavior first. First thing we noticed is that there is a good correlation between manual and automatic grading. Besides, the displayed results showed that the average grade for assessments manually marked are slightly higher than those graded automatically. To further examine this phenomenon, we have compared each pair of assessment (manual and automatic), individually as well as performed a detailed code analysis for each submission when there is a notable difference between the obtained marks.

Table 1. Comparison of automatic assessment and manual assessment results

	Manual Assessment			Automatic Assessment		
	Average Mark	Median	Standard Deviation	Average Mark	Median	Standard Deviation
Exercise 1	7.21	8	2.37	6.88	8	2.53
Exercise 2	6.84	7	2.23	6.29	7	2.85
Exercise 3	5.14	5.75	2.91	3.67	3.5	3.25

The following graphs summarize the grades of the three exercises both manually and automatically. In Figures 2, 3 and 4 student's submissions are presented in horizontal axis and grades (0–10) in vertical. The line graph in blue (with circular dots) displays the submissions' manually assessed grades whereas the red line graph (with square dots) represents the corresponding automatic assessment grades.

Although the similarities are much more present than the dissimilarities between the two assessment approaches, this section will explore factors that may contribute to grading discrepancies. First, we call a grading discrepancy a difference by at least 2 points between the manual and the automatic grade. In these cases, we have found that the program doesn't run correctly; do not provide the expected results in the dynamic analysis phase but found an approximate match in the solution space because close enough to a working solution. That confirms our hypothesis of not to be limited to the dynamic analysis alone in our grading system.

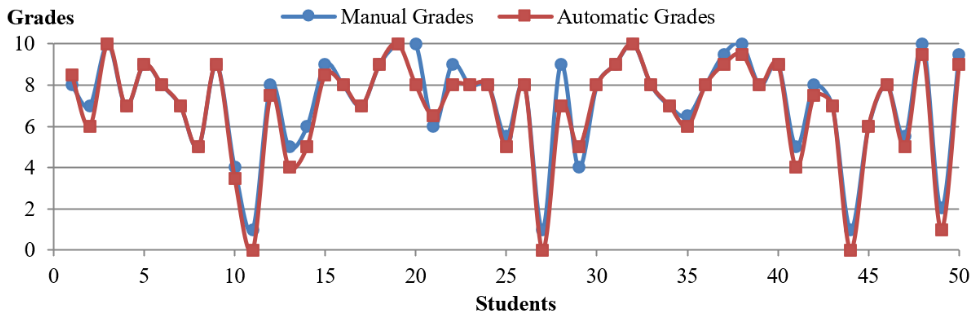

Figure 2. Exercise 1, manual versus automatic grades

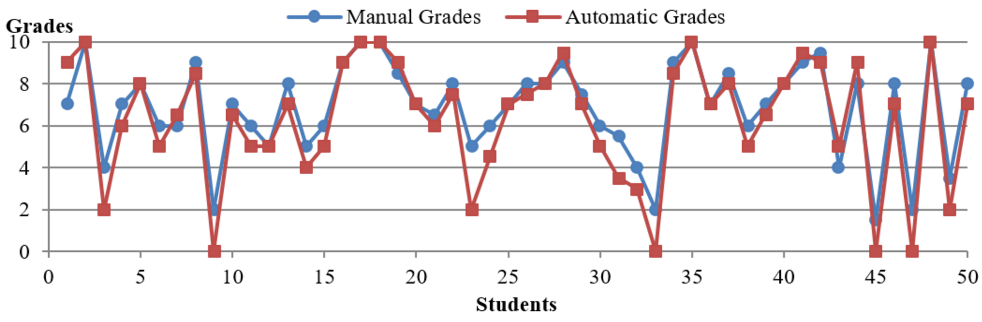

Figure 3. Exercise 2, manual versus automatic grades

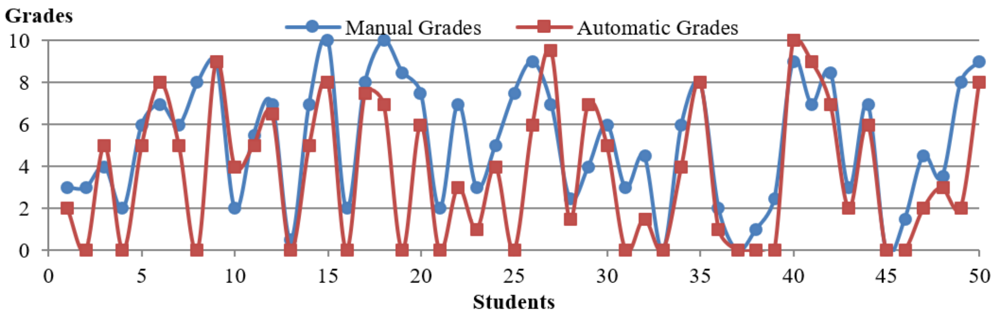

Figure 4. Exercise 3, manual versus automatic grades

Reviewing meticulously these cases, we have elaborated a list of the reasons why the programs failed during automated testing. One typical mistake we have encountered is buffer overflow errors. Another common mistake leading to a difference between the two grades was that the student program doesn't return a value. Another group of mistakes are programs with errors which occurred due to students' carelessness and rush: i) missing semicolon, ii) misspelled keywords or standard functions, iii) variables that are left uninitialized, and iv) missing case in a switch/case selection control, and v) undeclared variables.

As a matter of fact, these cases did not occur frequently in our three introductory C programming exercises. Even if the mistakes list seems large, it is due to the fact that some students' code displayed multiple errors. Another source of grades discrepancies is that some students' have handed uncompleted programs. These programs are given close to 0 points by the automated system. However, the teacher has decided to award them some intermediate mark, based on the knowledge and skills they demonstrated, instead of marking their copies 0.

Last, we have noticed some values in the three graphs where the instructor grade is way less than the system's and that's because the submitted programs even if correct didn't meet the functional requirement of the assignment. We could say that in general, when a program contains errors, the manual marking in this study appears to be much more permissive than the automatic marking.

In order to statistically validate the upper mentioned observations, we have performed a correlational analysis, which is a statistical technique that can show whether and how strongly the pairs of grades are close. Since the human and the automatic grades are not normally distributed, we opted for the Spearman's rank correlation coefficient on all three exercises. The result of computing this statistic will give a value that varies from +1 through 0 to –1, with 1 and −1 being the strongest positive and negative correlation, respectively. A value of 0 indicates that the values are not correlated at all. In addition, we measured the probability (p) of how likely or probable it is that any observed correlation is due to chance. These correlations, presented in Table 2, proved a strong significant relationship between both assessment methods:

We noticed some pronounced differences in the third exercise but without falling from the strong correlation range. This is, in part, explained by the increased difficulty of the third exercise compared with the first two. While analyzing out-of-range cases, we found that the main reason for the low precision was lacking tuned parameters. In fact, the teacher has been more lenient on some aspect of the third exercise probably due to the fact that fewer students were able to code it correctly. We can deal with these cases by providing the human assessor with access to the parameters. We remind that we didn't tune them, deliberately, to study their impact on the system scoring efficiency. Probably a stricter instructor would have met better the systems grading.

Table 2. Spearman's rank correlation coefficient

	Spearman coefficient (rs)	probability (p)
Exercise 1	0.94	< 0.001
Exercise 2	0.91	< 0.001
Exercise 3	0.72	< 0.001

5. CONCLUSION

The here presented system has been designed to be part of a submission environment with a goal to provide great benefits for both the student and the instructor. The main motivation was to obtain objective, efficient and fast assessment of programming assignments, in circumstances as plausible as increased students-instructor ratios and not so plausible as a pandemic forcing educators and students to shift to online learning and assessment. This system has the potential to improve the learning in the field of programming courses by making the assessment mechanism fast and simple to use. The general idea is first, to execute the student's code with a predetermined set of inputs then compare the outputs with the expected results. Afterwards, we measure the similarity of the student's program with a set of solutions predefined by the teacher for each exercise. Program graph representation, semantic-preserving transformations and program matching are used in this approach. Finally, we obtain a grade that reflects the dynamic and the static assessment results combined with two penalty parameters so that the predictive model's behavior matches the teacher's marking style and the exercise objectives.

To have a real estimation of the proposed system efficiency and reliability, an experimental study was carried out with some exams of a C-language introductory course. Through this experiment, two data sources were collected: the grades obtained by manually assessing students' programming assignments, and the grades marked automatically with our tool, for the same students' submissions. The reported results showed a strong correlation between the two sets of grades, validating our assessment system results. Nevertheless, there are a number of errors that students made, although minor, that caused the automated assessment system to be more severe in grading than the manual assessment. A measure that teachers can use to enhance students' understanding toward the grades awarded by these systems is to clarify how the assessment system works and urge the students to pay special attention to these types of mistakes. Another measure that could reduce the number of out-of-range cases is the tuning of the grading function parameters according to the teachers' goals and assessment styles.

In sum, the results revealed that the system had a normal grading behavior compared with grades awarded by instructors. However, our approach still has limitations. At present, it is only applied to simple introductory programs. Our current work includes improving the similarity-based grading approach, such as adding more standardization rules and dealing with complex programming exercises, such as structures and pointers. At the moment, we have limited the aforementioned system to work with a single programming language i.e., the C-language. This is done in order to detect eventual flaws and refine it as a first step. However, implementation doors have been kept open, so that we can extend the system to include programming assignments in other popular languages such as C++, Java, by making small changes like defining the compiler and the testing framework as arguments. In our future work we plan to study our system behavior compared with another automatic assessment tool for computer programs. This study should explore the utility and limitations of each approach, assess which one can improve students' achievement in programming and provide a considerable benefit in the field of automatic grading assessment of programs.

REFERENCES

[1] W. Pullan, S. Drew, and S. Tucker, "A problem based approach to teaching programming," in *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering FECS'13*, 2013, pp. 1–4.

[2] N. C. C. Brown and G. Wilson, "Ten quick tips for teaching programming," *PLOS Computational Biology*, vol. 14, no. 4, Apr. 2018, doi: 10.1371/journal.pcbi.1006023.

[3] R. Romli, S. Sulaiman, and K. Z. Zamli, "Automatic programming assessment and test data generation a review on its approaches," in *2010 International Symposium on Information Technology*, Jun. 2010, pp. 1186–1192, doi: 10.1109/ITSIM.2010.5561488.

[4] T. Staubitz, H. Klement, J. Renz, R. Teusner, and C. Meinel, "Towards practical programming exercises and automated assessment in massive open online courses," in *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, Dec. 2015, pp. 23–30, doi: 10.1109/TALE.2015.7386010.

[5] C. Lee and W. T. de Vries, "Sustaining a culture of excellence: massive open online course (MOOC) on land management," *Sustainability*, vol. 11, no. 12, Jun. 2019, doi: 10.3390/su11123280.

[6] E. Wetzinger, B. Standl, and G. Futschek, "Developing a MOOC on introductory programming as additional preparation course for CS freshmen," in *Proceeding of EdMedia + Innovate Learning*, 2018, pp. 1–10.

[7] S. Pokhrel and R. Chhetri, "A literature review on impact of COVID-19 pandemic on teaching and learning," *Higher Education for the Future*, vol. 8, no. 1, pp. 133–141, Jan. 2021, doi: 10.1177/2347631120983481.

[8] W. Admiraal, B. Huisman, and O. Pilli, "Assessment in massive open online courses," *Electronic Journal of e-Learning*, vol. 13, no. 4, pp. 207–216, 2015.

[9] J. Hollingsworth, "Automatic graders for programming classes," *Communications of the ACM*, vol. 3, no. 10, pp. 528–529, Oct. 1960, doi: 10.1145/367415.367422.

[10] C. Douce, D. Livingstone, and J. Orwell, "Automatic test-based assessment of programming," *Journal on Educational Resources in Computing*, vol. 5, no. 3, Sep. 2005, doi: 10.1145/1163405.1163409.

[11] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, Jun. 2005, doi: 10.1080/08993400500150747.

[12] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of recent systems for automatic assessment of programming assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research - Koli Calling '10*, 2010, pp. 86–93, doi: 10.1145/1930464.1930480.

[13] F. Restrepo-Calle, J. J. Ramírez Echeverry, and F. A. González, "Continuous assessment in a computer programming course supported by a software tool," *Computer Applications in Engineering Education*, vol. 27, no. 1, pp. 80–89, Jan. 2019, doi: 10.1002/cae.22058.

[14] D. M. Souza, K. R. Felizardo, and E. F. Barbosa, "A systematic literature review of assessment tools for programming assignments," in *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, Apr. 2016, pp. 147–156, doi: 10.1109/CSEET.2016.48.

[15] J. Caiza and J. Del Alamo, "Programming assignments automatic grading: review of tools and implementations," in *7th International Technology, Education and Development Conference (INTED2013)*, 2013, pp. 5691–5700.

[16] H. Keuning, J. Jeuring, and B. Heeren, "Towards a Systematic Review of Automated Feedback Generation for Programming Exercises," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, Jul. 2016, pp. 41–46, doi: 10.1145/2899415.2899422.

[17] P. Maguire, R. Maguire, and R. Kelly, "Using automatic machine assessment to teach computer programming," *Computer Science Education*, vol. 27, no. 3–4, pp. 197–214, Oct. 2017, doi: 10.1080/08993408.2018.1435113.

[18] A. Lajis, S. A. Baharudin, D. A. Kadir, N. M. Ralim, H. M. Nasir, and Normaziah Abdul Aziz, "A review of techniques in automatic programming assessment for practical skill test," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 10, no. 2–5, pp. 109–113, 2018.

[19] Z. Ullah, A. Lajis, M. Jamjoom, A. Altalhi, A. Al-Ghamdi, and F. Saleem, "The effect of automatic assessment on novice programming: Strengths and limitations of existing systems," *Computer Applications in Engineering Education*, vol. 26, no. 6, pp. 2328–2341, Nov. 2018, doi: 10.1002/cae.21974.

[20] F. A. D. O. Santos, "A tool for assisted correction of programming exercises in Java based in computational reflection," *Journal on Computational Thinking (JCThink)*, vol. 2, no. 1, Nov. 2018, doi: 10.14210/jcthink.v2.n1.p51.

[21] D. Galan, R. Heradio, H. Vargas, I. Abad, and J. A. Cerrada, "Automated assessment of computer programming practices: the 8-Years UNED experience," *IEEE Access*, vol. 7, pp. 130113–130119, 2019, doi: 10.1109/ACCESS.2019.2938391.

[22] T. Wang, X. Su, P. Ma, Y. Wang, and K. Wang, "Ability-training-oriented automated assessment in introductory programming course," *Computers & Education*, vol. 56, no. 1, pp. 220–226, Jan. 2011, doi: 10.1016/j.compedu.2010.08.003.

[23] S. H. Edwards and M. A. Perez-Quinones, "Web-CAT: automatically grading programming assignments," *ACM SIGCSE Bulletin*, vol. 40, no. 3, pp. 328–328, Aug. 2008, doi: 10.1145/1597849.1384371.

[24] M. Joy, N. Griffiths, and R. Boyatt, "The boss online submission and assessment system," *Journal on Educational Resources in Computing*, vol. 5, no. 3, Sep. 2005, doi: 10.1145/1163405.1163407.

[25] D. Fonte, I. V. Boas, D. da Cruz, A. L. Gancarski, and P. R. Henriques, "Program analysis and evaluation using quimera," in *Proceedings of the 14th International Conference on Enterprise Information Systems*, 2012, pp. 209–219, doi: 10.5220/0004001702090219.

[26] M. T. Pham and T. B. Nguyen, "The DOMjudge based online judge system with plagiarism detection," in *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, Mar. 2019, pp. 1–6, doi: 10.1109/RIVF.2019.8713763.

[27] S. Srikant and V. Aggarwal, "A system to grade computer programming skills using machine learning," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug. 2014, pp. 1887–1896, doi: 10.1145/2623330.2623377.

[28] A. Rump, A. Fehnker, and A. Mader, "Automated assessment of learning objectives in programming assignments," in *International Conference on Intelligent Tutoring Systems*, 2021, pp. 299–309, doi: 10.1007/978-3-030-80421-3_33.

[29] S. Zougari, M. Tanana, and A. Lyhyaoui, "Towards an automatic assessment system in introductory programming courses," in *2016 International Conference on Electrical and Information Technologies (ICEIT)*, May 2016, pp. 496–499, doi: 10.1109/EITech.2016.7519649.

[30] S. Gupta, "Automatic assessment of programming assignment," in *Computer Science & Engineering*, Jan. 2012, pp. 315–323, doi: 10.5121/csit.2012.2129.

[31] S. Zougari, M. Tanana, and A. Lyhyaoui, “Graph based hybrid assessment system for programming assignments,” *International Journal of Computer Applications*, vol. 178, no. 14, pp. 56–60, May 2019, doi: 10.5120/ijca2019918929.
[32] M. Nikolić, “Measuring similarity of graph nodes by neighbor matching,” *Intelligent Data Analysis*, vol. 16, no. 6, pp. 865–878, Nov. 2012, doi: 10.3233/IDA-2012-00556.
[33] G. Aman, “Best programming languages to learn in 2021 (for job & future),” *Hackr.io*, 2021.

BIOGRAPHIES OF AUTHORS

Soundous Zougari received her computer science engineering degree from the National School of Applied Sciences of Tangier (ENSA Tangier), University of Abdelmalek Essadi, Morocco and Ph.D. degree in computer science from University of Abdelmalek Essadi, Tetouan, Morocco, in 2021. Her research interests are in the areas of: learner’s assessment in eLearning, Intelligent Tutoring Systems for learning to program, heterogeneous distributed eLearning systems, eLearning standards and computer supported collaborative environments, and data structures. She can be contacted at email: soundous.zougari@gmail.com.

Mariam Tanana received the engineering degree in computer science from the National Graduate School of Engineering of Caen, France (ENSI), in 1990. She received her Ph.D. in computer science from the National Institute of Applied Sciences of Rouen, France (INSA), in 2009. Currently, she is a professor authorized to conduct research at the National School of Applied Sciences of Tangier (ENSA Tangier), University of Abdelmalek Essadi, Morocco. Her research interests include the learner’s assessment in e-Learning. She can be contacted at email: mtanana@uae.ac.ma.

Abdelouahid Lyhyaoui received the bachelor's degree in electrical engineering from University of Abdelmalek Essaadi, Tetouan, Morocco, in 1992, the M.S. signal system and radio communication degree from Escuela Técnica Superior de Telecomunicaciones, Universidad Politécnica of Madrid, in 1996, and the Ph.D. degree from Universidad Carlos III de Madrid, Spain, in 1999. Between 2000 and 2003 he was a Visiting Professor in Signal Theory and Communications at Universidad Carlos III de Madrid, Spain. Currently, he is a professor at National School of Applied Sciences of Tangier, Abdelmalek Essaadi University. His main research interests include statistical learning theory, neural networks and their applications in multimedia signal processing and education. He can be contacted at email: a.lyhyaoui@uae.ac.ma.