Bey, A., Jermann, P., & Dillenbourg, P. (2018). A Comparison between Two Automatic Assessment Approaches for Programming: An Empirical Study on MOOCs. *Educational Technology & Society, 21* (2), 259–272.

A Comparison between Two Automatic Assessment Approaches for Programming: An Empirical Study on MOOCs

Anis Bey[1*], Patrick Jermann[2] and Pierre Dillenbourg[3]

[1]Ecole Supérieure des Sciences de Gestion, LRI, Université Badji Mokhtar-Annaba, Algeria // [2]Center for Digital Education, Ecole Polytechnique Fédérale de Lausanne, Switzerland // [3]Computer Human Interaction in Learning and Instruction, Ecole Polytechnique Fédérale de Lausanne, Switzerland // anis.bey@univ-annaba.org // patrick.jermann@epfl.ch // pierre.dillenbourg@epfl.ch

[*]Corresponding author

ABSTRACT

Computer-graders have been in regular use in the context of MOOCs (Massive Open Online Courses). The automatic grading of programs presents an opportunity to assess and provide tailored feedback to large classes, while featuring at the same time a number of benefits like: immediate feedback, unlimited submissions, as well as low cost of feedback. The present paper compares Algo+, an automatic assessment tool for computer programs, to an automatic grader used in a MOOC course at EPFL (Ecole Polytechnique Fédérale de Lausanne, Switzerland). This empirical study explores the practicability and behavior of Algo+ and analyzes whether it can be used to evaluate a large scale of programs. Algo+ is a prototype based on a static analysis approach for automated assessment of algorithms where programs are not executed but analyzed by looking at their instructions. The second tool, EPFL grader, is used to grade programs submitted by students in MOOCs of Introductory programming with C++ at EPFL and is based on a compiler approach (Dynamic Analysis approach). In this technique submissions are assessed via a battery of unit tests where the student programs are run with standard input and assessed on whether they produced the correct output. In this study results showed the advantages and limits of each approach and pointed out how the two tools can be used to get a benefit assessment of students' learning in MOOCs of computer programming. This study led to the proposition of a model for the relationship between the number of submissions and the appearance of the most frequent submitted programs. This technique is used by Algo+ for giving feedback and it is based only on the *n* most redundant submissions that have been annotated by the instructor.

Keywords

Computer education, Programming assignments, MOOCs, Assessment, Automated grading, Static analysis, Dynamic analysis, CS1

Introduction

In introductory programming classes, teachers are often faced with grading a huge number of assignments. Because grading requires dealing with the complexities of the submissions, it becomes a strenuous and burdening task for teachers. To deal with this problem, there are generally two ways: increasing the number of teachers and the use of computers. The first approach is often not feasible (because of personnel costs) whereas using computers to run automatic assessment tools allows dealing with this issue. This approach has many advantages: objective assessment, removal of halo effects, and reduction in manual, error-prone and marking burden (Kalogeropoulos et al., 2011).

Computer based assessment is useful for handling very large numbers of students. Apart from giving a score, these tools may also offer an environment to practice programming, which is useful for students to develop programming skills. Since programming cannot be learned solely from books as in other subjects, students have to learn programming by developing algorithms themselves to deepen their understanding (Lahtinen et al., 2005).

In the last years, MOOCs (Massive Open Online Courses) represent the new form of learning and the new way to teach a large number of students without a place or time restriction. A massive number of students subscribe to these courses through several online platforms like EdX, Coursera and Udacity that give multiple opportunities to students to resubmit their code to improve on their mistakes, providing the opportunity for mastery learning. Since assessment resources are not restricted, automated grading allows students to increase mastery by iteratively improving and resubmitting their homework.

A big challenge that needs to be addressed for MOOCs to work is automated assessment of student work. The tremendous number of submissions that require feedback justifies the need for automated assessment tools. Given the popularity and widely claimed promise of MOOCs technology, we submit that how students are assessed and which forms this assessment takes are vital research questions. The involvement of a service for

automatic assessment of programs in a MOOC provides additional benefits such as formative assessment, repeatability of assessment and timely feedback; which constitute alternative pedagogical benefits. In general, the quality of assessment in MOOCs has been characterized by some authors such as in (Admiraal et al., 2014) and (Clara & Barbera, 2014). In MOOCs context, assessment and feedback are an integral part of the learning process and not only the final step in learning (Sandeen, 2013).

Another area of special interest is learning programming. In the last decades, programing is known as the 21st century skills. As programing skills become ever more important and a core competency for all kinds of 21st Century jobs, this is leading researchers to seek out new ways of learning to program. But learning and teaching the basics of programing is a complicated task and the best way for learning programing is practicing programing techniques and concepts by trying them out yourself (Lahtinen et al., 2005). Programming is not an exact science, but the more you practice, the more you develop skills; "practice makes perfect," and to do, students need an efficient assessment tool to tell them if they gave a correct or a wrong solution, and to give an efficient and timely feedback with a summative assessment to quantify achievement.

In the current study, we examined two automated graders: (1) Algo+, a prototype system developed to assess algorithmic competencies (Bey & Bensebaa, 2011; Bey & Bensebaa, 2013), based on static program analysis approach and (2) EPFL grader, an automated grader developed at EPFL (Ecole Polytechnique Fédérale de Lausanne, Switzerland) and based on dynamic program analysis. This tool is used to assess students' programs in the MOOC Introduction to Programming (with C++). And the question that guided this study was: Could Algo+ tool also be used to automatically assess the submissions of students in the context of MOOCs? To form a better view of the effectiveness of Algo+, we compared its results to those of EPFL grader real data and we examined the behavior of Algo+ tool in the case where it would be used in the context of MOOCs learning.

The rest of this paper is organized as follows. Primarily, we show the different forms of assessment used in MOOCs of computer programming, some related works and the main approaches used in automated assessment in programming. Also, we briefly state the functionalities of Algo+ and EPFL grader. The method design is presented with a description of data collected, results presentation and analysis of both scoring results and types of feedback provided by each tool. Finally, we conclude this study by proposing a model about the relationship between the number of submissions and the number of the appearance of the most frequent submissions.

Assessment in MOOCs of computer programming

Assessment covers an important part of MOOCs and constitutes the key driver for learning. The actual providers of MOOCs like Coursera and edX propose simple tools in the context of programming such as Multiple-Choice Questions and Peer Assessment. How to design assessments is a challenge in itself as MOOCs have massive and diverse student enrollment.

We can usually distinguish three manners to assess computer programming skills in MOOCs. The first uses simply the online assessment through multiple choice questions or multiple choice cloze tests. This kind of assessment is easy to be automated but it is not suitable for assessing conceptual skills in computer programming.

The second form is the Peer Assessment. Self and peer assessment might offer promising solutions that can scale the grading of complex assignments in courses with thousands of students. This form of assessment has been historically used for logistical, pedagogical, metacognitive, and affective benefits. For example, Coursera uses peer grading by submitting an assignment and five people grade it; in turn, you grade five assignments. In general, students becoming assessors may bring multiple spin-offs. It can be a significant learning experience if it was well explained and designed to the students and on the other hand, a huge number of submissions can be graded. But leaving assessment in the hands of the students can generate a depreciated assessment because they may create their own criteria that may not be well founded.

The third way is using a third-party grader tool option. The use of automatic graders is not typical in most of MOOCs providers in their programming courses. They let MOOC's owners use external graders to provide interactive, dynamic, online coding exercises and more complex programming assignments for learners. The execution of codes must be totally independent as illustrated in Figure 1 to avoid that malicious codes or badly written codes affect the MOOC system.

In general, the choice of technique for the assessment in MOOCs for programming depends on what instructors want to assess in students after following the MOOC and what they want to know about their skills. But it is essential to mention that the assessment process has to take into account some features like assessing problems having different answers or partially correct answers options.

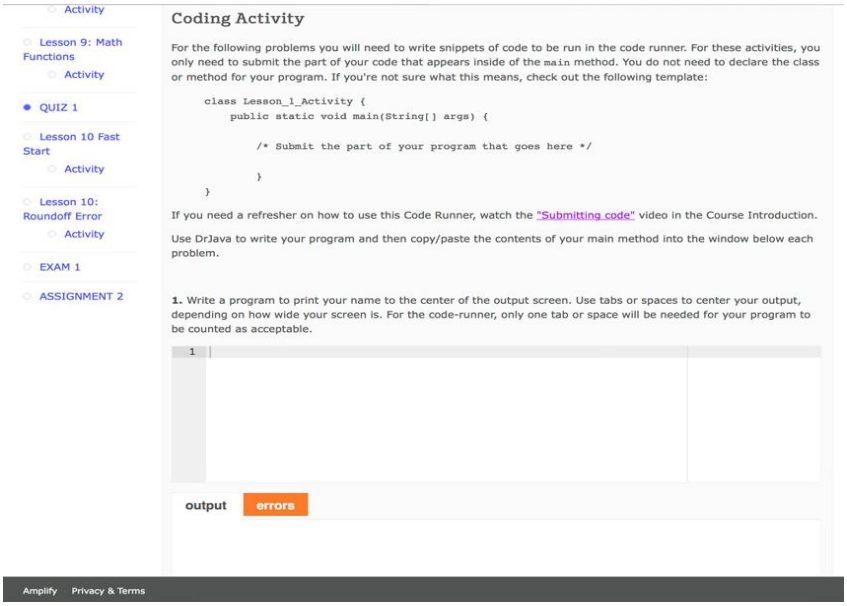

Figure 1. Submission page of a MOOC of programming using a submitting space for writing code

Related works

Automated assessment tools in programming has often been studied since 1960 by Hollingsworth (Hollingsworth, 1960) and are growing in popularity with the appearance of MOOCs to deal with the colossal number of submissions.

We do not provide an exhaustive review of the existent automated grading systems in this paper. Various studies have investigated the value of automated assessment tools and surveys. Ala-Mutka (2005) and Ihantola et al. (2010) listed a comprehensive overview of those systems. We can cite some systems and tools that can be found in almost all literature review studies on automated assessment in computer programming: CAP (Schorsch, 1995), Ceilidh (Benford et al., 1995), ASSYST (Jackson, 2000), CourseMarker (Higgins et al., 2003), TRAKLA2 (Korhonen et al., 2003), Curator (Edwards, 2003), Submit (Harris et al., 2004), TRAKLA (Malmi et al., 2005), BOSS (Joy et al., 2005), Mooshak (Guerreiro & Georgouli, 2006), Autograder (Nordquist, 2007), WebCAT (Edwards & Perez-Quinones, 2008), Athene (Towell & Reeves, 2009), AutoLep (Wang et al., 2011), JUG (Brown et al., 2012), Aari (Taherkhani et al., 2012), COALA (Jurado et al., 2012), CodeWrite (Denny et al., 2014).

WebCAT (Edwards & Perez-Quinones, 2008) is one of the most popular automated assessment tool used by many institutes to assess students' programs source code. This tool provides some useful features (i) it allows submissions directly from within Eclipse using a simple plug-in (also from a few other IDEs, or directly via a web browser), (ii) instructors can use JUnit tests to check student solutions; Checkstyle and PMD to perform static analysis checks on student solutions, (iii) allows students to write and submit their own JUnit tests. Of course, WebCAT is based only on Java programming language.

To date, there are two main approaches used for grading students' programs. The first approach is dynamic analysis. It runs a program through a data set and then compares the output to the predefined answer. The second approach does not run student programs but it looks for metrics; such as line of code, number of variables, statements and expressions, or attempts to compare the source code of the student program with a model program.

Although many automated programming assessment systems have been proved to be of great help to both instructors and students in learning programming but several problems remain unsolved. Foremost among these problems are security, algorithms for automatic generation of test data in dynamic analysis, low accuracy and

precision of correctness and functionality assessment in static analysis. However, assessing the functionality of students' code is still the most often used approach to grade programs due to its simplicity of implementation by running programs.

Algo+ and EPFL grader overviews

EPFL grader

The EPFL grader is an automated grader used to assess students' assignments in programming during the MOOCs course Introduction to programming with C++ (a MOOC presented by Jamila Sam, Jean-Cédric Chappelier et Vincent Lepetit through Coursera platform in Ecole Polytechnique Fédérale de Lausanne, Switzerland). For every session, this course provides a set of programming exercises for participants. Exercises can be solved by submitting a source code as a file within the web browser (see Figure 2). Solutions submitted by students are compiled and unit-tested over a set of inputs. In return, students receive a score and an automatic feedback on how their code performed in the tests. There is no limit to submitting solution programs.

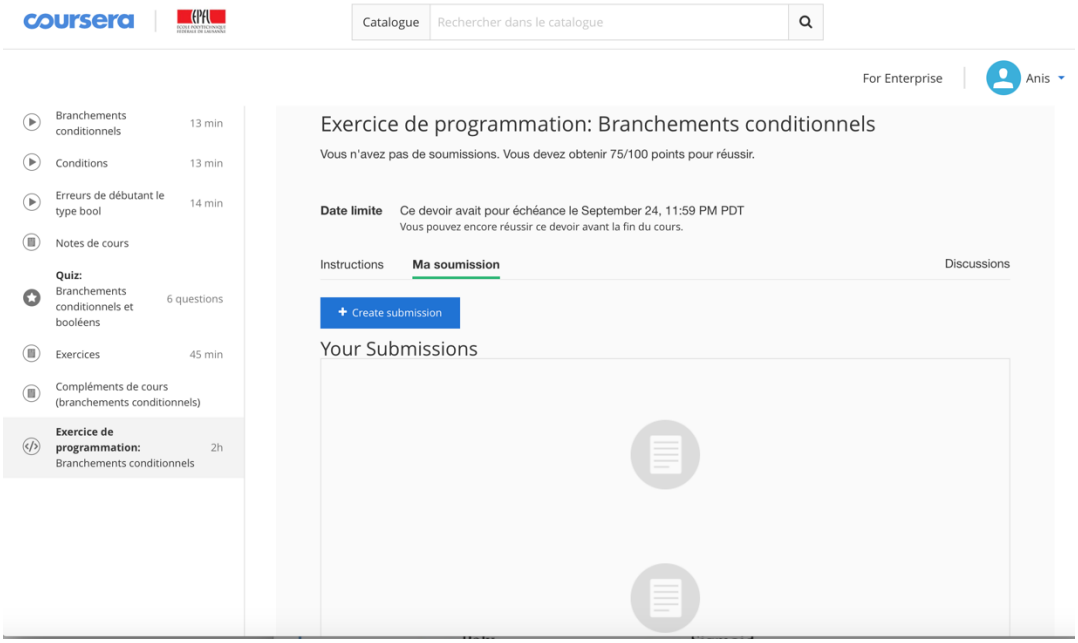

Figure 2. Submission page of Introduction to programming with C++ MOOC at EPFL using the EPFL grader

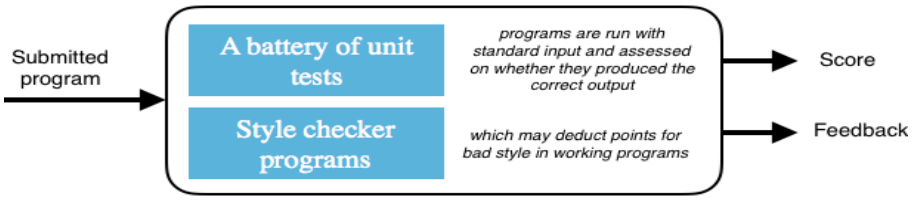

Figure 3. EPFL grader mechanism

As illustrated in Figure 3, the automated grader of EPFL programming MOOC is based on two main components: (1) a battery of unit tests where students' programs are run with standard input and assessed on whether they produced the correct output, and (2) a style checker which may deduct points for bad style in working programs and give feedback. Students are able to experiment with the tool and find out for themselves whether something works or not within a feedback.

Algo+

Algo+ is an automated assessment tool of programs using program matching (Bey & Bensebaa, 2011; Bey & Bensebaa, 2013). A submitted program is assessed by comparing it to a set of predefined solutions already assessed by an instructor. Predefined solutions are called *Referent Solutions* and are those common and frequent

submissions that have been detected in students' submissions. The set of referent solutions used to assess submitted programs contains not only correct programs but also erroneous ones. The whole solutions (correct and erroneous) are gathered and used as a source of learning and assessment for the new submitted cases (see Figure 4).

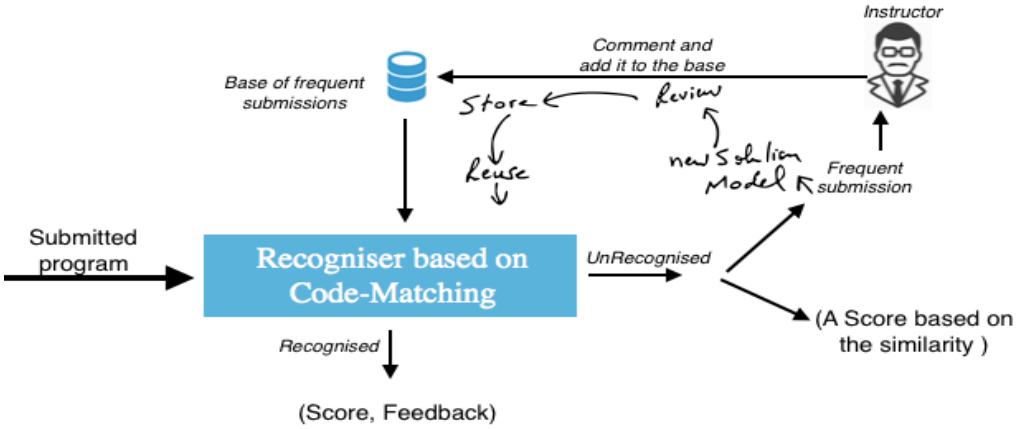

Figure 4. Algo+ mechanism

Each referent solution in the database has a score and a feedback giving by the instructor. The feedback giving by the instructor for each referent solution should be general. It explains misconceptions, if the case of erroneous solution, or general explanation about how the solution is made if it is correct.

Algorithm 1: Algo+ algorithm for assessing students' programs

```
Input  : RS[nb_RS]; /* list of Referent Solutions
         RE[nb_RE]; /* list of Referent Errors
         Submission;
Output: Score, feedback1, feedback2
Sim_max ← Dist(Submission, RS[0]);
Ref← RS[0];
for i ← 1 to nb_RS do
    Sim ← Dist(Submission, RS[i]);
    if Sim_max < Sim then
        Sim_max←Sim;
        Ref←RS[i];
    end
end
if Sim_max = 1 then
    Score←Ref.Score;
    feedback1←Ref.feedback;
else
    Score←Ref.Score * Sim_max;
    Sim_max←Dist(Submission, RE[0]);
    Ref←RE[0];
    for i ← 1 to nb_RE do
        Sim←Dist(Submission, RE[i]);
        if Sim_max < Sim then
            Sim_max←Sim;
            Ref←RE[i];
        end
    end
    if Sim_max > 0.5 then
        feedback2←Ref.feedback;
    end
end
write(Score,feedback1,feedback2);
```

Algorithm 1: Algo+'s algorithm describing steps for grading students' programs

Figure 4 illustrates how a submitted program is assessed. The recognizer module is based on code-matching and it described in the algorithm (see Algorithm 1). Correct referent solutions are used for scoring while erroneous solutions are used to giving feedback. It tries to recognize the submitted program among referent solutions. If the submitted program was recognized among referent solutions, the score and the feedback of the referent solution are returned to the student. Otherwise, the submitted program is considered as unrecognized and stored in frequent submission database. If it appeared more than two times from different students then it would be considered as common and sent to the instructor which has to award an appropriate score and feedback. The score attributed to student in this case is calculated from the similarity value.

Study design

This empirical study was undertaken on real datasets. The comparison was performed with large samples of programs collected from a MOOC course in Introduction to programming with C++ at EPFL. It is one of the important MOOCs courses provided by EPFL. Approximately, 13500 participants follow this MOOC.

The research question asked in this study is: How well could Algo+ assess students' submissions in a MOOC of programming compared to EPFL grader?

Data collection

In this work, we use datasets collected from a course of Introduction to programming with C++ taught by EPFL professors. All collected submissions were submitted between September 2013 and September 2014. In this course, a grading system is used to assess students' submissions. Students are asked to download a file containing a pre-established source code where it is indicated to put the asked algorithm according to the statement of the exercise.

Two different exercises were selected for this study. The first exercise (Exercise 1) was about swapping values. Students were required to write a C++ program that swap three values. For example, for these input data a = 51, b = 876 and c = 235, we obtain a = 235, b = 51 and c = 876.

The second exercise (Exercise 2) was to write a C++ program that guesses which character (among a list known in advance) the user has in mind. The purpose of this exercise is the ability to use conditional structure.

Fortunately, an electronic archive of all submissions made during the session within the assessment results provided by EPFL grader (submission ID, mark and date) are available. A total of 2312 participants completed Exercise 1 and produced 4985 submissions. Only 3130 (n = 3130) programs are used in this study after eliminating programs that are damaged or considered not valuable. It is the case when the code was inserted into the inappropriate place where it has been indicated in the pre-established source code. Otherwise, the submitted program will not be assessed and a value of 0 is automatically attributed. In the second exercise 1991 participants produced 7067 programs and solely 4914 programs (n = 4914) were considered in this experiment.

On Algo+ side we need at least one referent solution to be able to assess submitted programs. In this study, we have started with two referent solutions for each exercise. Scores assigned by the two tools are rated on [0-30] for the first exercise and [0-50] for the second exercise.

Results

This study was designed to determine whether Algo+ can assess students' submissions in MOOCs as well as EPFL grader. The obtained results were reported according to three important features that an automated assessment tool has to have in the context of MOOCs: correctness (summative), usefulness of the feedback (formative) and the speed of assessment (massive).

The main summary descriptive statistics are presented in Table 1. Both measure of center that is mean and median of EPFL grader and Algo+ are almost similar in the first exercise but different in the second exercise. The measure of variation that is standard deviation is also similar in the first exercise but different in the second. The means and medians of Algo+ and EPFL grader scores were (M = 20.26, SD = 14.03) and (M = 21.6, SD =

12.06) in Exercise 1 and (M = 24.76, SD = 22.2) and (M = 28.96, SD = 17.02) in Exercise 2. In addition, the Wilcoxon signed rank test was performed to examine the range of score frequencies and showed that there is a significant difference in the range of score frequencies ($p < .05$).

Table 1. Descriptive statistics

	Exercise 1		Exercise 2	
	Algo+	EPFL grader	Algo+	EPFL grader
Min	0.00	0.00	0.00	0.00
1st quartile	0.00	10	0.00	16.35
Median	30.00	30.00	30	29.81
Mean	20.26	21.6	24.76	28.96
3rd quartile	30.00	30.00	50	50
Max.	30.00	30.00	50	50
SD	14.03	12.06	22.20	17.02
Mode	30	30	0	50

Results of correlational analyzes, using the nonparametric Spearman Rank Correlation Coefficient tests, indicated that statistically significant correlations were present between Algo+ and EPFL grader in both exercises (Exercise 1: rs = 0.92, $p < .001$ and Exercise 2: rs = 0.59, $p < .001$). These results show excellent correspondence between the two sets of marks for both the direct comparison with EPFL grader and with the ranked order of student programs in the two exercises.

To check if there are differences in scores distributions between Algo+ and EPFL grader marks, we have presented in Figure 5 histograms of scores of both Algo+ and EPFL grader in the two exercises. We compared scores distribution using two-sample Kolmogorov-Smirnov tests, which found significant differences for the two exercises ($p < .05$). In the two exercises in question, we can easily distinguish that scores assigned by EPFL grader were even more concentrated on two values in the first exercise (the minimum score 0 and the largest score 30) and six values in the second exercise (0;10;20;30;40;50). Contrary to Algo+ where scores distribution is concentrated on values ranged between 0-30 and 0-50 for the first and the second exercise, respectively.

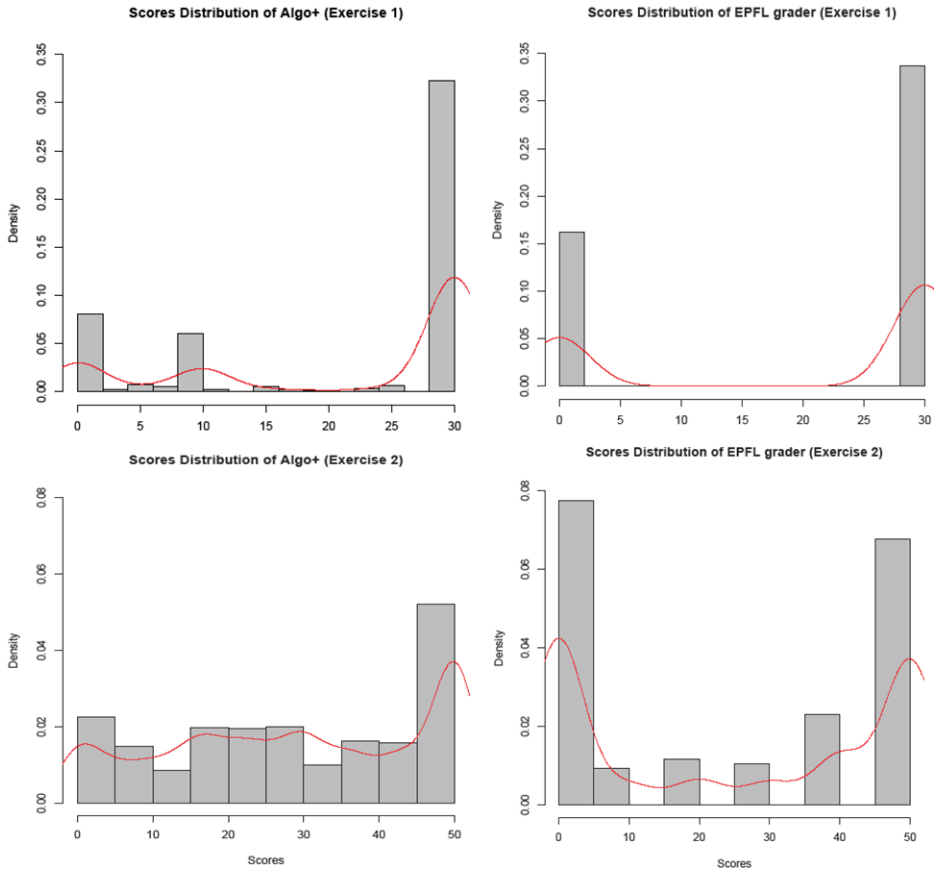

Figure 5. Histograms showing scores distribution of Algo+ and EPFL grader

As scores assigned by Algo+ and EPFL grader are rational numbers (e.g., 2.3; 2.4; 2.5) it is difficult and not very meaningful to calculate kappa or the percentage of absolute agreement. Under these conditions, the inter-rater agreement between Algo+ and EPFL grader was calculated with the intraclass correlation coefficient (ICC) (McGraw & Wong, 1996) with a two-way mixed model using R version 3.2.4. The intraclass correlation (ICC) is a measure of agreement and it is useful when there are many rating categories (5 or more) or when ratings are made along a continuous scale. The "agreement" ICC is the ratio of the subject variance by the sum of the subject variance, the rater variance and the residual. The inter-rater agreement between Algo+ and EPFL grader was substantial in the first exercise (ICC = 0.88, 95% CI 0.86-0.89, $p < .001$) but it was a moderate agreement in the second exercise (ICC = 0.51, 95% CI 0.48-0.55, $p < .001$) based on the cut-off value for acceptability level (Graham et al., 2012).

To further investigate scores assigned by the two tools and understand differences, frequencies were performed to examine the distribution of scores. And for ease of comparison and discussion, we presented the frequency tables by three score ranges, namely, by erroneous programs (scores= 0), somewhat correct programs (scores between 0 and the largest score) and correct programs (scores equal to the largest score).

Table 2 and Table 3 present data on similarities and differences in scores between EPFL grader and Algo+ according to these three score ranges (correct, somewhat correct and erroneous).

Table 2. Concordance and discordance between Algo+ and EPFL grader in Exercise 1 (n = 3130)

		Algo+			
		Correct	Somewhat-Correct	Erroneous	Total
EPFL grader	Correct	2022	38 (A)	39 (B)	2099
	Somewhat-Correct	0	14	5 (C)	19
	Erroneous	0	557 (D)	455	1012
	Total	2022	609	499	3130

Table 3. Concordance and discordance between Algo+ and EPFL grader in Exercise 2 (n = 4914)

		Algo+			
		Correct	Somewhat-Correct	Erroneous	Total
EPFL grader	Correct	1242	49 (A)	362 (B)	1653
	Somewhat-Correct	0	209	1696 (C)	1905
	Erroneous	0	136 (D)	1220	1356
	Total	1242	394	3278	4914

Note. (A) (B) (C) (D) are types of disagreement between Algo+ and EPFL grader (illustrated in Table 4).

The two graders agree on scores for 2490 (79%) cases in the first exercise and 2671 (54%) in the second exercise but differ in the rest of cases (21% and 46% for Exercise 1 and Exercise 2, respectively).

On the other hand, three types of differences were identified:

- *Case 1:* Submissions judged correct by EPFL grader but somewhat correct or erroneous by Algo+. This case represented 77 (2%) and 411 (8%) submissions in the first and the second exercise, respectively *(disagreement type A and B illustrated in Table 4).*
- *Case 2: Submissions judged* somewhat correct by EPFL grader but erroneous by Algo+. This case represented 5 (0.1%) and 1696 (34%) cases in exercise 1 and exercise 2, respectively *(disagreement type C illustrated in Table 4).*
- *Case 3: Submissions judged* erroneous by EPFL grader but somewhat correct by Algo+. This case represented 557 (18%) and 136 (2%) submissions in the exercise 1 and exercise 2, respectively *(disagreement type D illustrated in Table 4).*

To illustrate each case of disagreement between Algo+ and EPFL grader, we have chosen some illustrative samples from the two exercises in Table 4. According to these examples, we can highlight that EPFL grader awards high scores than Algo+ when the submitted program is not frequent because Algo+ does not recognize it among referent solutions. In the other case, when Algo+ awards high scores than EPFL grader, is due to the fact that Algo+ awards scores even if the submitted program does not give a correct output.

Table 4. Some examples that illustrate differences in scoring between Algo+ and EPFL grader

Disagreement	Exercise1	Exercise2	Comments
A	int tmp = 0; tmp = c; c=b; b=tmp; tmp = b; b=a; a=tmp; *(EPFL grader=30; Algo+=12.5)*	if (chapeau) { if ((homme) && (!moustaches) && (lunettes)) { cout << "le Professeur Violet";}else { cout << "Il n'existe pas";}}else { if (moustaches){ if ((homme) && (lunettes)) { cout << "le Colonel Moutarde";}else { cout << "Il n'existe pas";}}else { if (lunettes) { if (homme) { cout << "le Reverend Olive";}else { cout << "Mme Leblanc";}} else { cout << "Mlle Rose »;}}} *(EPFL grader=50; Algo+=20)*	Correct programs that were not frequent and thus Algo+ did not assigned the total score
B	swap(a,b); swap(b,c); swap(a,b); swap(c,b); *(EPFL grader=30; Algo+=0)*	if (homme && lunettes) { cout << "le Professeur Violet"; }else {cout << "erreur";}} else { if (homme) { if (lunettes) { if (moustaches) { cout << "le Colonel Moutarde"; }else {cout << "le Reverend Olive";}}else { cout << "erreur";}}else { if (lunettes) { cout << "Mme Leblanc";}else { cout << "Mlle Rose";}}} *(EPFL grader=50; Algo+=0)*	Correct programs that were not frequent and are not similar to any referent solution. Therefore Algo+ did not recognize this submission and assigned the lowest score (0)
C	a=b+a; b=a+(-b); c=(b+a)/2; *(EPFL grader=10; Algo+=0)*	if (chapeau) {} if (moustaches){ if(lunettes){ if(chapeau){} else{ if(homme){cout << "le Colonel Moutarde";}}}}else { if(lunettes) {if(chapeau){if(homme){cout << "le Professeur Violet" ;}}else{ if(homme){cout << "le Révérend Olive";}else{ cout << "Mme Leblanc";}}}else{ if(chapeau){}else{ if(homme){}else { cout << "Mlle Rose »;}}} *(EPFL grader=40; Algo+=0)*	Somewhat correct programs that were not frequent, consequently, Algo+ did not recognize them
D	int nTemp ; nTemp = c ; b = a ; c = b ; a = nTemp ;	if (chapeau) { cout << "le Professeur Violet";} else if (moustaches) { cout << "le Colonel Moutarde";} else if (not lunettes) { cout << "Mlle Rose";} else if (homme) { cout << "le Révérend Olive";} else {cout << "Mme Leblanc";}	Programs that did not produce correct output (EPFL grader=0) but they have a part of correctness in their code. Algo+ awards an average score according to how much the program is similar to a referent correct program

(EPFL grader=0; Algo+=23.25) *(EPFL grader=0; Algo+=37.5)*

Analyze

The correlations between Algo+ and EPFL grader were calculated using Spearman's rho and showed a high correlation between the two systems in the first exercise (rs = 0.92, $p < .001$) and less high correlation in the second exercise (rs = 0.59, $p < .001$) compared to the first exercise. Overall, there was a positive correlation between Algo+ and EPFL grader scores.

When the two distributions scores of the two exercises were compared, we found that the high agreement in the first exercise (ICC = 0.88, 95% CI 0.86-0.89, $p < .001$) is due to the simplicity of the exercise where variability of solutions is less important (72 distinct correct solutions produced by students where only 33 submissions were frequent) compared to the second exercise (ICC = 0.51, 95% CI 0.48-0.55, $p < .001$) where students came up with 482 correct ways to solve the exercise with 119 were frequent. The disagreement between EPFL grader and Algo+ is mainly explained by the fact that scores assigned by Algo+ is based solely on the frequent correct solutions. The rest of correct solutions that are not frequent, Algo+ will not recognize them and thus awards low scores.

However, when we examined the ranges of marks we found similarities and differences. Algo+ and EPFL grader assess correct programs with the same accuracy but differently when programs are not correct. Results showed that Algo+ and EPFL grader assess using different assessment metrics. Algo+ assessment is based on the existence of each part of the program in the referent solution whereas EPFL grader on the output correctness. Consequently Algo+ attributes high scores compared to EPFL grader especially in the case of erroneous programs. In this case Algo+ gives some points for approximate programs even if they do not run. On the other side, EPFL grader does not award any points when the program does not run or it does not produce the correct outputs.

In other words, EPFL grader does not assess bad programs, it assesses only correct programs. It looks for the correctness of outputs while students can produce interesting programs but may they forgot some details due to carelessness or silly errors that make the program not-running or produce the wrong outputs.

Also, we analyzed the approach adopted by Algo+ that uses the *n* most frequent solutions to assess students' submissions. We have found that Algo+ has yieled coverage of 96% and 75% of students' submissions in the first and second exercise, respectively. We analyzed the rest of the not covered submissions and we have found only 2.3% and 7.5% of submissions in the first and second exercise, respectively, that represent correct and not frequent submissions, and thus Algo+ did not assess them correctly. These isolated cases come from some students that proposed functionally correct but somewhat weird solutions.

Types of feedback

One of the key challenges of program grading is designing feedback to the student to help them understand defects in their program (Wilcox, 2016). Both Algo+ and EPFL grader provide a feedback when they assess students' submissions. By the nature of the assessment (static and dynamic program analysis) several differences were found to be significant.

Algo+ provides feedback with two parts. The first part of feedback is the annotations of the instructor about the conception of the whole solution. This feedback is annotated by the instructor when a submission has been added into the referent solutions database. If the referent solution is correct, a general explanation of how it is structured. Likewise, if the solution contains a misconception and semantic errors, also an explanation of this misconception is presented for student. The second part of feedback is generated from the difference between the submitted program and the most similar correct referent solution. This difference shows student how s/he has to carry out to obtain the correct solution. It shows students the missed instructions and which modifications have to be performed to get a valid solution. Using the feedback on the most *popular* erroneous programs tries to help students that do not understand why their programs do not provide the correct answers even if they are syntactically correct.

However, the feedback provided by EPFL grader is based on check style process and test cases. The generated feedback informs student about the test case achieved and not achieved by presenting the output of the submitted program and the expected output. Moreover, penalties on each undesirable practice such as unread variable are also generated by the feedback.

Speed and quality of assessment of Algo+

As Algo+ assessment is based on the most popular submissions and needs instructor intervention to annotate them, we have tried to explore the notion of reaching performance - we want to know how many submissions all frequent solutions could appear to let Algo+ independent from instructor and hence ensure its efficiency. We have chosen the number of submissions as an independent variable rather than the time spent between submissions because the number of submissions can change one in a while during a MOOC session.

In this study, Algo+ started assessment session with two programs that represent referent solutions in each exercise. During submitting programs (Exercise 1, n = 3130 and Exercise 2, n = 4914) to Algo+, instructor has manually assessed 33 and 119 programs in Exercise1 and Exercise 2 respectively. These programs were appeared twice and more during the MOOC session and that have been assessed by the instructor.

To analyze the mechanism of Algo+ in reaching performance and hence to determine how many times instructor will be solicited in each exercise, we have passed to Algo+ the same chronological submissions of EPFL grader real session and we calculated the number of frequent solutions in each of 50 passed submissions.

After the examination of the relationship between the number of submissions and the number of frequent submissions that can appear, we have found that this relation is linear at the beginning but after some number of submissions, the number of frequent programs becomes rapidly constant and reaches an asymptotic level. This is a characteristic of hyperbolic models. What they have in common is a specified maximum value of the dependent variable, and a description of when the maximum value is attained in respect of the independent variable. There are several different types of hyperbolic functions. Among those models, the Michaelis-Menten model of substrate uptake. It consists of studies of the growth rates of microbes under controlled conditions commonly examine the relation between some measure of growth rate, and the substrate concentration. In the following section, we explained how the Michaelis-Menten model is used and adapted in our context.

Toward modeling the number of frequent submissions in MOOCs based on adapted enzyme kinetics model

Algo+ adopts an approach of giving feedback based on the n most common submissions that instructor has to annotate. An automated feedback is giving by Algo+ when redundant submissions were detected. Redundant submissions are submissions that have the same syntactic structure. In this section, we attempted to conceive and to develop a model to accomplish the goal of estimating the number of frequent submissions (the n most common solutions) for a given exercise.

To fit a model, the functional form of the growth trajectory of the frequent submissions being modeled must be chosen. To select the appropriate functional form for our model, we identified a small number of model types that aligned well with our data (i.e., they featured an interpretable upper asymptote to estimate the number of frequent submissions) and then tested their fit to the data empirically. Specifically, we compared the empirical fit of three different models with differing functional forms including S-shaped and J-shaped growth trajectories and identified the Michaelis-Menten model as the best fitting. The Michaelis-Menten model posits a J-shaped growth line and was first formulated in the field of biochemistry to estimate the rate of enzyme reactions based on the concentration of a substrate (English et al., 2006).

Notationally, the Michaelis-Menten growth trajectory of the number of frequent submissions according to the total number of submissions can be written as:

$$N = N_0 + \frac{N_{max} submissions}{K + submissions} \qquad (1)$$

In Equation 1, the three parameters of the Michaelis-Menten function are: the initial value (*N_0*), which is the initial number of solution defined by the instructor when the MOOC session does not started yet (submissions =

0); the rate parameter (*K*), which represents the point in submissions when the number of frequent submissions is halfway between the initial value (*N_0*) and the asymptote; and the asymptote (*N_max*), which characterizes the maximum value of the frequent submissions as the number of submissions approaches infinity.

Figure 6 shows the growth of the number of referent submissions during each number of submissions and the fitted line using the adapted MM model.

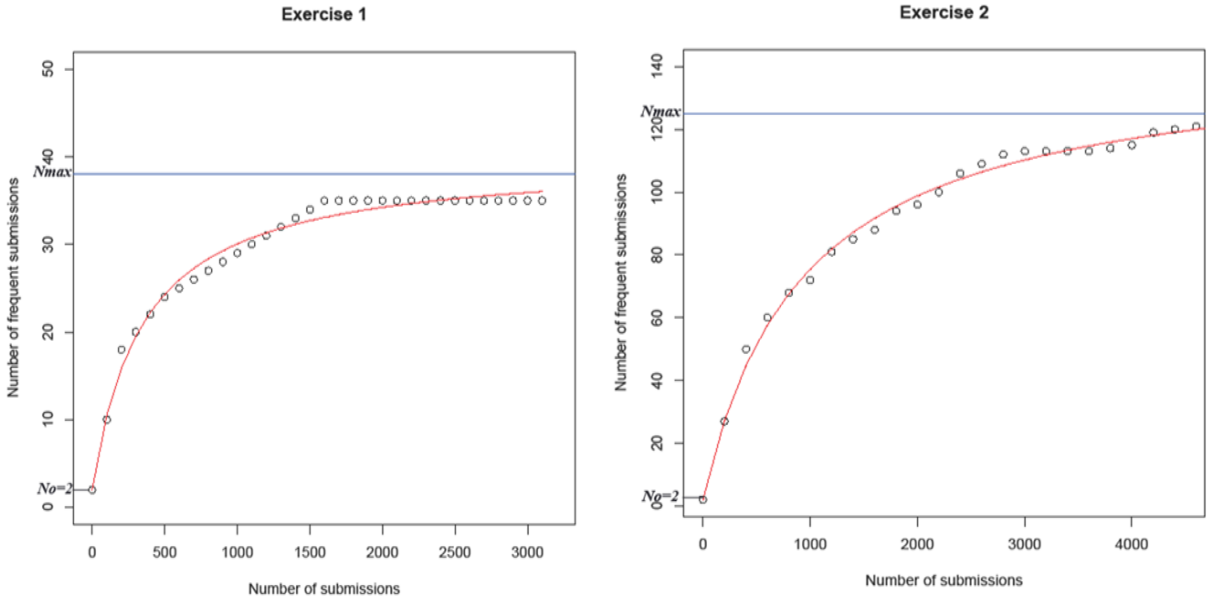

Figure 6. Plot of the number of frequent submissions (N) detected by Algo+ in the two exercises and the fitted line using the adapted Michaelis-Menten model. Submissions are chronologically ordered

Conclusions and future work

In this work, Algo+'s behavior was examined when placed in the context of MOOC by comparing it to EPFL grader experience. Despite the difference of the adopted assessment approach by each tool, a significant correlation existed between the two tools. Scoring results have shown that the EPFL grader awards scores according to the functionality of the submitted program while Algo+ awards scores according to the existence of correct codes even if the program does not run.

Regarding the relation between Algo+ and EPFL grader that we can notice through this study, is that both can complement each other very well. We can use EPFL grader to assess the functionality of the submitted code and Algo+ to assess non-functional programs.

In the light of these study, we have also explored the utility and limitations of the supervised approach used by Algo+ where instructor has to assign feedback and score only on the n most frequent submissions. This conducted us to propose a model about the relationship between the number of submissions and the number of frequent submissions that can appear in a given programming assignment during a MOOC session. The capitalization of the *n* most frequent submissions used by Algo+ for assessing is a process meant to build up a capital from the huge number of submissions in a MOOC session, in order to develop the ability of learning programming by making (knowledge and competencies) available to other institutions and students.

For future work, we still need to assess how much Algo+ can improve students' achievement in programming and the efficiency of the given feedback. This study has opened up new research areas for further improvements and future work. An automated grader that combined the two approaches can be developed. Also, it would be worthwhile to test the proposed model with different exercises to test the efficiency of predicting the critical number of submissions required to achieve the asymptotic level of frequent submissions and hence we could know, in an exercise, after how many submissions the *n* most frequent submissions will appear.

Acknowledgments

We thank Ian Anthony Flitman and Jean-Cédric Chappelier from EPFL for their support and for providing us the EPFL grader data.

References

Admiraal, W., Huisman, B., & Van de Ven, M. (2014). Self and peer assessment in massive open online courses. *International Journal of Higher Education*, *3*(3), 119-128.

Ala-Mutka, K. M. (2005). A Survey of automated assessment approaches for programming assignments. *Computer Science Education*, *15*(2), 83-102.

Benford, S. D., Burke, E. K., Foxley, E., & Higgins, C. A. (1995). The Ceilidh system for the automatic grading of students on programming courses. In *Proceedings of the 33rd annual on Southeast regional conference* (pp. 176-182). Clemson, South Carolina: ACM.

Bey, A., & Bensebaa, T. (2011). Algo+, an assessment tool for algorithmic competencies. In *IEEE Global Engineering Education Conference, EDUCON'11* (pp. 941-946). Amman, Jordan: IEEE.

Bey, A., & Bensebaa, T. (2013). Assessment makes perfect: Improving student's algorithmic problem-solving skills using plan-based programme understanding approach. *International Journal of Innovation and Learning*, *14*(2), 162-176.

Brown, C., Pastel, R., Siever, B., & Earnest, J. (2012). JUG: A JUnit generation, time complexity analysis and reporting tool to streamline grading. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12* (pp. 99-104). Haifa, Israel: ACM.

Clara, M., & Barbera, E. (2013). Three problems with the connectivist conception of learning. *Journal of Computer Assisted Learning*, *30*(3), 197-206.

Denny, P., Luxton-Reilly, A., & Carpenter, D. (2014). Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 conference on Innovation & technology in computer science education -ITiCSE '14* (pp. 273-278). Uppsala, Sweden: ACM.

Edwards, S. (2003). Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing*, *3*(3), 1-24.

Edwards, S., & Perez-Quinones, M. (2008). Web-CAT: Automatically grading programming assignments. *ACM SIGCSE Bulletin*, *40*(3), 328-328.

English, B., Min, W., Van Oijen, A., Lee, K., Luo, G., Sun, H., Cherayil, B., Kou, S., & Xie, X. (2006). Ever-fluctuating single enzyme molecules: Michaelis-Menten equation revisited. *Nature Chemical Biology*, *2*(3), 168-168.

Graham, M., Milanowski, A., & Miller, J. (2012). *Measuring and promoting inter-rater agreement of teacher and principal performance ratings.* Washington, DC: Center for Educator Compensation Reform. Retrieved from http://files.eric.ed.gov/fulltext/ED532068.pdf

Guerreiro, P., & Georgouli, K. (2006). Combating anonymousness in populous CS1 and CS2 courses. *ACM SIGCSE Bulletin*, *38*(3), 8-12.

Harris, J. A., Adams, E. S., & Harris, N. L. (2004). Making program grading easier: but not totally automatic. *Journal of Computing Sciences in Colleges*, *20*(1), 248-261.

Higgins, C., Hegazy, T., Symeonidis, P., & Tsintsifas, A. (2003). The CourseMarker CBA system: Improvements over Ceilidh. *Education and Information Technologies*, 8(3), 287-304.

Hollingsworth, J. (1960). Automatic graders for programming classes. *Communications of the ACM*, *3*(10), 528-529.

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). 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* (pp. 86–93). Koli, Finland: ACM.

Jackson, D. (2000). A Semi-automated approach to online assessment. *ACM SIGCSE Bulletin*, *32*(3), 164-167.

Joy, M., Griffiths, N., & Boyatt, R. (2005). The Boss online submission and assessment system. *Journal on Educational Resources in Computing*, *5*(3), Article 2.

Jurado, F., Redondo, M., & Ortega, M. (2012). Using fuzzy logic applied to software metrics and test cases to assess programming assignments and give advice. *Journal of Network and Computer Applications*, *35*(2), 695-712.

Kalogeropoulos, N., Tzigounakis, I., Pavlatou, E., & Boudouvis, A. (2011). Computer-based assessment of student performance in programing courses. *Computer Applications in Engineering Education*, *21*(4), 671-683.

Korhonen, A., Malmi, L., & Silvasti, P. (2003). TRAKLA2: A Framework for automatically assessed visual algorithm simulation exercises. In Proceedings of Third Annual Baltic Conference on Computer Science Education (pp. 48–56). Joensuu, Finland: University of Joensuu and University of Helsinki.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. (2005). A Study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, *37*(3), 14-18.

Malmi, L., Karavirta, V., Korhonen, A., & Nikander, J. (2005). Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *Journal on Educational Resources in Computing*, *5*(3), 1-23.

McGraw, K., & Wong, S. (1996). Forming inferences about some intraclass correlation coefficients. *Psychological Methods*, *1*(1), 30-46.

Nordquist, P. (2007). Providing accurate and timely feedback by automatically grading student programming labs. *Journal of Computing Sciences in Colleges, 23*(2), 16-23.

Sandeen, C. (2013). Assessment's place in the new MOOC world. *Research and Practice in assessment*, *8*, 5-12.

Schorsch, T. (1995). CAP: An Automated self-assessment tool to check Pascal programs for syntax, logic and style errors. *ACM SIGCSE Bulletin*, *27*(1), 168-172.

Taherkhani, A., Korhonen, A., & Malmi, L. (2012). Automatic recognition of students' sorting algorithm implementations in a data structures and algorithms course. In *The 12th Koli Calling International Conference on Computing Education Research-Koli Calling '12* (pp. 83-92). Koli, Finland: ACM.

Towell, D., & Reeves, B. (2010). From walls to steps: Using online automatic homework checking tools to improve learning in introductory programming courses. *ACET Journal of Computer Education and Research*, *6*(1), 1-8.

Wang, T., Su, X., Ma, P., Wang, Y., & Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course. *Computers & Education*, *56*(1), 220-226.

Wilcox, C. (2016). Testing strategies for the automated grading of student programs. In *Proceedings of the SIGCSE '16* (pp. 437-442). Memphis, TN: ACM.