Rym Aiouni* and Anis Bey
Ecole Supérieure des Sciences de Gestion-Annaba,
Laboratory of Research in Computer Science,
University of Badji Mokhtar,
Annaba, Algeria
Email: aiouni.rym@gmail.com
Email: anis.bey@univ-annaba.org
Email: beyanisse@gmail.com
*Corresponding author

Tahar Bensebaa
Department of Computer Science,
Laboratory of Research in Computer Science,
University of Badji Mokhtar,
Annaba, Algeria
Email: tahar.bensebaa@univ-annaba.org

Abstract: Programming skills become ever more important and a core competency in 21st century and for its learning students need learning environments where they can practice and receive immediate assessment that help them to progress. In this paper we have developed eALGO, an automated assessment tool of flowchart programs that allows the students to practice algorithms problems. The assessment method used in eALGO is based on graph matching. To validate the method, an experimental study was conducted for investigating the effects of the automated flowchart algorithms scoring system compared to teachers scoring. About 35 flowchart algorithms were selected during a lab session and scored by eALGO system and after that scored by five teachers. Statistical analysis of the results reveals that eALGO tool seems to correlate well with teachers. Furthermore, a strong correlation was noticed between eALGO and the average scores of teachers. Hence, the findings of this study show that using automated assessment of flowcharts algorithms based on graph matching methods can help teachers to alleviate the scoring load, while allowing students learning designing algorithms.

Keywords: computer-based assessment; automated assessment; programming; graph matching; similarity; novice programmers; flowchart; computer science education.

Biographical notes: Rym Aiouni is an Assistant Professor of Computer Science at Ecole Supérieure des Sciences de Gestion-Annaba. She prepared a PhD thesis at the Laboratory of Research in Computer Science in University of Badji Mokhtar, Annaba, Algeria. Her PhD research is computer-based assessment in programming. Her main research is technology enhanced learning.

Anis Bey is currently an Associate Professor at the Ecole Supérieure des Sciences de Gestion-Annaba and a member in the research group of Technology Enhanced Learning at the Laboratory of Research in Computer Science in the University of Badji Mokhtar, Annaba. He obtained his PhD thesis in 2013 from the University of Badji Mokhtar-Annaba, Algeria. His current research projects focus on automated assessment in programming, MOOCs and learning analytics.

Tahar Bensebaa is a Professor of Computer Science at the University of Badji Mokhtar-Annaba, Algeria since 1992. He obtained his PhD thesis in 1991 from INSA Lyon, France. He is the Head of the research group in Technology Enhanced Learning at the Laboratory Research in Computer Science. He is interested in hypermedia, collaborative learning, pedagogical simulation, computer assisted instruction, CSCL and e-learning.

1 Introduction

Programming is a discipline used for a long time in a naive way with no particular formalism. It has always been problematic, no particular formalism. This discipline is often source of problems for both the teacher and the student. For the teacher, because he has to find the right methods to help students assimilate abstract concepts. On the other hand, for students who are still in their initiation phase, the problem is even more important. It has been noticed that the abandon or failure rate in introductory courses in programming for freshmen (undergraduate) range from 25% to 80% (Kaasboll, 2002). According to some cognitive psychological studies, this is mainly due to the nature of the taught discipline. These studies have identified the major axes of the intrinsic difficulties of algorithms. In Algorithms, unlike other sciences such as physics, the student does not have a simple model viable of computer, which could serve him as a base to build more sophisticated models. In the contrary, his experience with it seems to favour an anthropomorphic model which does not allow him to understand the brutal return of error faced at the beginning of his practice of programming.

Another specific algorithmic difficulty is the abstraction of the task. The learner must factorise in the algorithm, the set of behaviours of the task. The result is a ‘blank page syndrome’, highlighted in particular by Kaasboll (2002). This raises the key question: what teaching methods proposed and with what tools can we improve learning programming? In recent years, the integration of information and communication technologies (ICT) has revived the improvement of the quality of teaching and learning different skills. We argue that the appropriate use of ICT with innovative teaching methods and tools appropriate to the context could be the solution to the problem of learning algorithms (Amerein et al., 1998; Benabbou and Hanoune, 2007).

The technology enhanced learning (TEL) has known substantial improvement efforts. For instance, they have been formed that formalisms are needed whether in the way to
describe, to index pedagogical contents or to script educational activities. In this sense, the evaluation of algorithms for learning using ICT has known a very important achievement in terms of developing new tools for automatic assessment as was stated in Higgins et al. (2005).

The main difficulty associated with this state of facts, just the assessment itself. Assessment in classroom was always the mysterious task (Charle, 1997). Several methods and tools have been devoted to the evaluation but they all suffer from failure. This inefficiency is due either to doubtful results uniqueness; that is to say, they cannot be applied to any area (we cannot evaluate the algorithmic skills with filling the gaps method).

Furthermore, the algorithmic assessment activity is among the most burdensome because algorithms are characterised by the multitude of solutions to a given problem. This feature increases the difficulty of evaluation in learning systems: experts find it hard to anticipate all the possible solutions to a problem to integrate them in the basic solutions (Guibert et al., 2005). Localisation of errors, which is an important factor in the progression of learners, is another difficulty resulting from this feature. This complicates the implementation of these systems.

Since long time, many experienced teachers in many universities have been in spite of their experience confronted to difficulties of their students face the problem of assessment of programming assignment. For this reason, a lot of software have been developed on the last decade. But unfortunately they suffer for many restrictions as inefficiency, restriction in designing programs, etc and do not fit well with the pedagogical requirement of the assessment task as any software development process (Wang, 2011).

In this paper, we have developed an automatic assessment system to assess students’ algorithms for learning. eALGO is the proposed tool to assess students’ flowchart programs using a graph matching method with predefined solutions. This proposal is made in the context of investigation of different automatic methods to be used in assessment of students’ algorithms initiated by Bey and Bensebaa (2013). The primary research question in the current paper was whether or not we obtain similar results of automatic assessment comparing to human expert assessment using flowchart representation of algorithms and graph matching method.

In Section 2 we display a brief description of some automatic assessment systems that adapt the use of flowchart and visual representation of programs in their pedagogical strategy. In Section 3 the procedure of assessment adapted by eALGO is depicted and we achieve this paper by discussing the results of the experimental study of eALGO with conclusions and the expected future works.

2 Existing assessment tools of programming

As programming skills become ever more important and a core competency in 21st century for almost all countries, this is leading individuals to seek out new ways of learning to program. Over the last decade interest has been rapidly growing in proposing automatic assessment tools of programming and many literature reviews appeared.

Among these studies that presented by AlaMutka where some of those systems and tools are presented. Ihantola et al. (2010) displayed automatic assessment of programming assignment. Where students write code and submit it for assessment.
Therefore, the rest of systems and tools based on visualisation approaches have not been included. As the proposed tool is based on flowchart representation of algorithms, we found that it is worthwhile to report a brief overview of existing visualisation tools in programming education. RAPTOR the programming flowchart-based environment is designed mainly to assist students in facing syntactic bugs and visualise the algorithms for them (Thomas et al., 2002).

The RAPTOR system aids the student in executing programs visually and tracing the execution with following flowcharts. Most students prefer to use flowchart to declare their algorithms, and results show that RAPTOR helps them more than traditional language or writing flowcharts without RAPTOR. The Raptor environment also allows users to design an algorithm by using flowchart signs and combine them together. Users can run the algorithm and see the process step by step or in continuous mode. Raptor uses pseudo code for editing flowchart and after that it converts it into Pascal program.

The Jeliot tool is conceived to assist novice in learning object oriented and procedural programming (Moreno and Myller, 2003). The idea behind Jeliot is to encourage students to construct their own programs and give them permission to see the visual illustration of the program execution. These processes help them to develop their mental model about the programming process that assists them in understand the concept of programming.

The iconic programming tool (B#) is a tool for beginner programmers (Greyling, 2006). The significant aim of design B# refers to difficulties faced in preliminary programming courses by novices, including problem-solving techniques and strategies, misconception about constructed programming languages, concepts and the traditional programming environments. It usually tries to make things easier in programming tasks by decreasing the level of accuracy and manual typing in programming languages. Thus, B# has provided an environment which aids students in programming by using flowcharts. Automatic generating codes, debugging and program executing are supported by the system as well. In this system there is too much focus on syntax, not sufficient emphasis on problem-solving and absence of support for experiencing program execution.

Structured flow chart (SFC) editor is a structured flowchart editor by Watts (2004). SFC allows the user to develop a structured flowchart, and always displays a pseudo code representation of the flowchart in either a C++ or Pascal-like syntax. The user then copies and pastes the textual representation into a text editor or integrated development environment (IDE) and makes changes to get a complete program.

Progranimate is the environment of the tool which is provided by the flowcharts, coding, tracing or allocating the variables (Scott et al., 2008). In some cases, the environment is very user friendly; however it is very simple in the first glance. This kind of tool can be very helpful to students, because the novice students can understand what is going on in the program without leaving the current page, and all the environment features are located in one place, which makes the tool more understandable. In fact, novices’ difficulty in programming related to points that they do not understand-especially where and how the variable initialised. If they able to see the process of the programming and tracing the code, that can be much easier to understand the concept. Progranimate uses Java and visual basic languages in its programming process.

Graphical representation of programs, the visualisation of data structures and variables as well as program animation are among the features that help to offer a more clear and visual display of the source code from the program in execution but they do not
tell students about the correctness of their solutions. We can clearly state that the most of tools do not pedagogically assess students’ flowcharts. We can summarise that systems that used flowchart representation are usually aimed at learning building programs and showing how it does work unlike systems dedicated to assess students’ programs as it has been reported in the study of Ihantola et al. (2010).

3 eALGO

eALGO is a flowchart-based programming tool designed specifically to help students create their algorithms and avoid syntactic baggage. eALGO algorithms are created visually but not executed and then assessed using static analysis through a graph matching process. Figure 1 shows the main components of eALGO.

Figure 1 Architecture of eALGO

Student can solve each algorithmic exercise by designing a solution using a dedicated flowchart editor, once ended it can be submitted for assessment by the matching process for assessment module. This one is parameterised beforehand by the instructor. If the submitted flowchart was recognised a result is sent to the student, else, the flowchart of the student is sent to the instructor for an assessment by hand and after that updating database if necessary.

The following two subsections explain why the flowchart representation has been chosen for expressing solutions and how it is created in eALGO. The next subsection shows the way in which the matching process for assessment does work.

3.1 Flowchart representation in eALGO

Unlike to the representation used in the work presented by Bey and Bensebaa (2013), which is shorthand notation for programming which uses a combination of informal programming structures and verbal descriptions of code, we have adapted a graphical representation using flowchart.
Flowcharts are a visual representation of program flow. A flowchart normally uses a combination of blocks and arrows to represent actions and sequence. Blocks typically represent actions. The order in which actions occur is shown using arrows that point from statement to statement. Sometimes a block will have multiple arrows coming out of it, representing a step where a decision must be made about which path to follow. This graphical representation was chosen for its many benefits. Flowcharts are better way of communicating the logic of an algorithm. Also, problem can be analysed in more effective way and thus makes program modification easier for learners.

It has been a long time since flowcharts have been used to visualise the structure of programs. The flowcharts are quite easy to understand without having any background in programming – with using algorithm beside flowcharts, the level of understanding will increase. Westphal says that “without the use of diagrams or flowcharts, it is difficult for beginners, even with pseudo code to communicate the flow of a program” (Westphal et al., 2003). Ben-Bassat says assuming that dynamic animation can expand the flowchart’s effectiveness as a novice assist in algorithmic problem solving and program development (Levy et al., 2001).

Research shows that “most students are visual learners and instructors tend to present information verbally; Studies estimate that between 75% and 83% of students are visual learners” (Fowler et al., 2000; Thomas et al., 2002).

### 3.1.1 Designing flowcharts in eALGO

In eALGO we create a flowchart using two types of actions. Elementary operation and basic operation denoted respectively by EO and BO. An EO is a simple action known in programming and that can be directly executed by the machine $EO = \{\text{write, read, test, assign}\}$. A BO is an operation that represents known algorithm such as Sorting, Deleting, Swapping, etc. With those operations, student can use control structures tests and loops for controlling.

Also and during creating the flowchart, student can use a third kind of operation as an intermediate phase. The role of these operations is to allow student to create gradually the solution and avoiding putting the whole operations together at the same time and so being unmanageable. The number of decomposition stages depends on the complexity of the problem; more the problem is complex, more the number of steps is important.

This method of successive refining processes (also called top-down approach) changes gradually and with maximum chances of success of the abstract description of the solution of the problem (for a complex operation) to the algorithm that will resolve it. The flowchart is at its last level when it contains only basic and EOs with eventual control structures.

This approach prevents the student from drawing in the details from the start and gradually decreases the complexity of the problem being addressed. Our goal with this approach is to evaluate algorithmic solutions. However, it is important not to overlook the essential fallout that constitutes learning by students of decomposition. Indeed, it is a must for the student in the formulation of the solution.

“A deep understanding of programming, in particular the notions of successive decomposition as a mode of analysis and debugging of trial solutions, results in significant educational benefits in many domains of discourse, including those unrelated to computers and information technology per se.” (Seymour Papert, in ‘Mindstorms’)
3.2 Assessment process

When a learner expresses his solution as a flowchart we try to assess this last one by comparing it with solutions predefined by the expert. So we can summarise the whole process according to three principal components as depicted in Figure 2.

**Figure 2** The process of matching process for assessment

3.2.1 Descriptor generator

To automate the comparison of the process of learning with those of the expert (solutions plan), we have been inspired by the work of Sorlin et al. (2006) on the measurement of multi-labelled graphs. This approach allowed us to propose a method for matching algorithmic solutions.

From the organisational structure of the learner, a description of the solution is generated. For this, we assigned to each operation and each transition a set of labels. The set of couple \((\text{Num}_\_\text{operation}, \text{label})\) and triples \((\text{Num}_\_\text{operationS}, \text{Num}_\_\text{operationT}, \text{label})\) are descriptors and constitute the description of the solution. The couple \((\text{Num}_\_\text{operation}, \text{label})\) describes operations of the flowchart and the triples \((\text{Num}_\_\text{operationS}, \text{Num}_\_\text{operationT}, \text{label})\) transition between operation (\(S\) and \(T\) are for source and target operation).

Given a set \(L_o\) of operation labels and a set \(L_t\) of edge labels, a flowchart algorithm is defined by a triple \(S =< O, \ ro, \ rt >\) such that:

- \(O\) is a finite set of operations
- \(\ro \subseteq O \times L_o\) is a relation associating labels to operations (edges in the flowchart), i.e., \(\ro\) is the set of couples \((opi, l)\) such that the operation \(opi\) is labelled by \(l\). For each edge of the flowchart, two descriptors are assigned, one contains the nature of the edge, if it is an operation, a test or a loop and the second contains the label. For
example in Figure 3, for the operation \textit{read}\((t[i])\) two descriptors were assigned; (1:\textit{EO}) to mention the type of the operation and (1:read\((t[i])\)) for labelling.

- \(rt \subseteq O \times O \times Lt\) is a relation associating labels to edges, i.e., \(rt\) is the set of triples \((opi, opj, l)\) such that edge \((opi, opj)\) is labelled by \(l\). Label in transition may take two values: \textit{seq} for sequence transition or true/false when it is an alternative test.

The description of the process \(S\) is the set of all its operations characteristics and transitions \(\text{Desc}(S) = ro \cup rt\).

### 3.2.2 Filtering

The filtering step is installed before the matching process in order to optimise the process of searching the most similar flowchart. It consists to find among the predefined flowcharts those who contain all the critical BOs that were prescribed by the expert. The filtering allows to decrease the number of solution to be matched. As a result, we obtain a subset of solutions from the predefined ones that contains critical operations. This subset of solutions will be presented as candidates where we try to find the closest one to the solution of the learner by the matching mechanism.

### 3.2.3 Matching process

A match between two steps \(S1 = \langle O1, ro1, rt1 \rangle\) and \(S2 = \langle O2, ro2, rt2 \rangle\), is a relationship: \(m \subseteq O1 \times O2\). Such matching associates each operation of the solution with the operation of the same order of the other solution.

To measure the similarity between two approaches with respect to the pairing \(m\), we suggest to adapt the formula of similarity (Tversky, 1977) generalised by Sorlin et al. (2006):

\[
\text{Sim}(S1, S2) = \frac{f\left(\text{descr}(S1) \cap \text{descr}(S2)\right)}{f\left(\text{descr}(S1) \cup \text{descr}(S2)\right)}
\]

Formula (1) calculates the similarity of two solutions, by matching their descriptors. The function \(f\) defines the relative importance of descriptors, with respect to each other. This function formula (2) is often defined as a weighted sum:

\[
f(F) = \sum_{(o1, o2, l) \in F} \text{Weight}(o1, l) + \sum_{(o1, o2, l) \in S} \text{Weight}(o1, o2, l)
\]

The assignment of weights to the various descriptors of a solution is performed by the expert. This weight reflects the importance of the descriptor in terms of the purpose of the exercise and what should be assessed by this exercise.

The method for determining a mark first computes a similarity measure (a value between 0 and 1) between relationships in the specimen solution and relationships in a student’s answer. Then, the best match is found the match between relationships which maximises the overall similarity between flowchart algorithms. The best match is then scored according to the given mark scheme provided by the expert.

Figure 3 demonstrates a detailed example of a student trying to represent a bubble sort in a flowchart. The example shows the matching process between a correct flowchart
(S_base) and a flowchart drawn by a student (S_student) who omits a loop. Two cases are considered in this example when the instructor assigns or not the flowchart to show how important weight assignment is to the final score.

**Figure 3** An illustrative example of a matching process between a student's and instructor's solution about bubble sorting algorithm

The first step is transforming the two flowcharts to descriptors by the generator. Next, we calculate similarity between the two sets of descriptors.

In the first case, when the instructor does not assign weights for the different operations of the flowchart, all descriptors are equal to 1, which means that all operations have the same importance and the instructor want to assess the existence of all operations in the flowchart with the same degree of importance.

In the second case, when the instructor assigns weight for an operation (the gray loop in this example) for its importance, this will decrease the similarity value and thus the score.
4 Experiment

The goal of the study is to measure the effectiveness of the system by comparing it against human experts’ assessment results. For this, we have conducted a lab study. This section describes the details of the results.

4.1 Method

The study involved 35 first year students of computer science and five faculty teachers of algorithms and data structures. During a lab session, students have been asked to produce flowcharts programs using eALGO system that solves an exercise about implementing mathematical function. The purpose of the given exercise is the ability to use conditional structure.

The participants’ flowcharts algorithms were scored instantly by eALGO and then retrieved by the researcher. The retrieved flowcharts algorithms were then printed and scored by five computer science teachers. These five computer science teachers did not proctor the test, nor did they teach this group of participants. Results of the automated scores were compared with those obtained manually by teachers. In section below, we describe the obtained results.

4.2 Results

This study was designed to determine whether eALGO marks closely match teachers, thereby validating its scoring method. In the experiment, the student flowchart algorithms were marked by eALGO and teachers working with a specimen solution illustrated in Figure 4.

Figure 4  The specimen solution used in the experimentation study (see online version for colours)
The descriptive statistics given in Table 1 show a strong homogeneity in assigning scores by eALGO as well as teachers. The mean marks show good agreement with the average ($M_{eALGO} = 2.22$, $M_{avg} = 1.98$) and with most of teachers ($M_{T1} = 2.21$, $M_{T3} = 2$, $M_{T4} = 2.57$). When we look at the correlation between the two sets of marks in Table 2, Pearson’s correlation coefficient is 0.67 (which is significant at the 0.001 level). Spearman’s rho statistic can be used to see how well the automatic marker ranks the students compared to the human markers and we obtain 0.74 (which is significant at the 0.001 level). These results show excellent correspondence between the two sets of marks for both the direct comparison with human marks and with the ranked order of student answers.

Table 1  Descriptive statistics

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>Average score</th>
<th>eALGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>1.00</td>
<td>0.50</td>
<td>0.75</td>
<td>1.00</td>
<td>0.50</td>
<td>1.15</td>
<td>0.70</td>
</tr>
<tr>
<td>First quartile</td>
<td>2.00</td>
<td>0.87</td>
<td>1.75</td>
<td>2.00</td>
<td>1.25</td>
<td>1.65</td>
<td>1.64</td>
</tr>
<tr>
<td>Median</td>
<td>2.25</td>
<td>1.00</td>
<td>2.00</td>
<td>2.50</td>
<td>1.50</td>
<td>1.95</td>
<td>2.47</td>
</tr>
<tr>
<td>Mean</td>
<td>2.21</td>
<td>1.34</td>
<td>2.00</td>
<td>2.57</td>
<td>1.78</td>
<td>1.98</td>
<td>2.22</td>
</tr>
<tr>
<td>Third quartile</td>
<td>2.50</td>
<td>1.75</td>
<td>2.25</td>
<td>3.00</td>
<td>2.50</td>
<td>2.27</td>
<td>2.88</td>
</tr>
<tr>
<td>Max.</td>
<td>3.25</td>
<td>4.00</td>
<td>3.25</td>
<td>3.50</td>
<td>3.40</td>
<td>3.40</td>
<td>3.26</td>
</tr>
<tr>
<td>Sd</td>
<td>0.53</td>
<td>0.73</td>
<td>0.59</td>
<td>0.75</td>
<td>0.85</td>
<td>0.49</td>
<td>0.77</td>
</tr>
<tr>
<td>Mode</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1.5</td>
<td>2.25</td>
<td>3.25</td>
</tr>
<tr>
<td>Variance</td>
<td>0.29</td>
<td>0.53</td>
<td>0.34</td>
<td>0.57</td>
<td>0.72</td>
<td>0.24</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 2  Correlation statistics

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>r(33)</td>
<td>0.51***</td>
<td>0.32*</td>
<td>0.47**</td>
<td>0.65***</td>
<td>0.44**</td>
<td>0.67***</td>
</tr>
<tr>
<td>p</td>
<td>0.53***</td>
<td>0.40**</td>
<td>0.48**</td>
<td>0.70***</td>
<td>0.49**</td>
<td>0.75***</td>
</tr>
</tbody>
</table>

Note: *p < .05, **p < .01, ***p < .001; avg. = average over all raters.

Figure 5  Distribution scores of automatic and human raters
According to Table 2 we note that all correlations are positive which means that all marks vary to the same sense. We found that eALGO correlate positively with all teachers and especially with T4 and the average over all teachers.

To highlight the distribution of scores, Figure 5 summarises the different scores of each teacher with eALGO.

4.3 Discussion and analysis

The purpose of this study was to consider the relationship between eALGO and human scoring in order to determine and to analyse the behaviour of eALGO for assessment flowchart algorithms. At the current stage of this research, feasibility study on using graph matching in assessing computer programs is concerned.

A correlational research design was used to answer the research question of this work, correlation between eALGO performance and teacher raters’ performance were examined.

The results of correlational analyses, using the non-parametric Spearman rank correlation coefficient tests, indicated that statistically significant correlations were present between the eALGO overall scores and average scores of teachers ($r_s = .75$, $p < .001$).

On the other hand, results have demonstrated a strong relationship between human average scores and automated scores of eALGO, thus implying the validity of computer generated scores. Also, results showed a positive correlation between raters and eALGO.

Figure 6  Principal component analysis between eALGO and teachers (see online version for colours)

This study has demonstrated that eALGO assess students’ algorithm independently of any idiosyncratic criteria contrary to human teachers that scoring is usually influenced by many facts as halo effect. And the most important finding is that eALGO tends to assign
scores with high agreement with not solely human raters average scores \( r(33) = 0.67, p < 0.001 \) but also with human raters one by one (Table 2).

For more detailed examination of the scoring logic of all teachers as well as the automatic scoring system eALGO, we have used principal component analysis to highlight similarities and differences between automatic and hand-scoring methods and to discover and summarise intercorrelations between human raters and eALGO.

Figure 6 shows that teachers and eALGO are oriented into the same direction which means that both teachers and eALGO assessed the same objects by the same way. The first principal component is strongly correlated with the six original variables.

The first principal component increases with increasing the automatic and human raters. This suggests that these six raters vary together. If one increases, then the remaining also increases their scores. This component can be viewed as the mean of the six raters. In fact, we could state that all correlations of the first principal component is between 0.37 and 0.47 which means that all raters correlates most strongly with this principal component that is the average mean scores.

The second principal component is strongly correlated with teacher number 5 that assigns low scores \( \text{median} = 1.5 \). This component can be viewed as a measure of severity or leniency. It increases with increasing severity and decreases with decreasing leniency of raters in assigning scores. In this exercise, eALGO could be seen as a lenient rater compared to teacher T2 and T5 and the most important find is that eALGO assessment cannot be differentiate between human raters.

5 Conclusions and future works

In this research, we have developed an automated scoring tool called eALGO based on program matching method offering to novice programmers an environment of practice and to have an instant correction of their solutions. In summary, the analysis of the investigation results discussed in this paper seems to suggest that automated grading using graph matching may safely reduce the burden of grading students especially in the case of large number of assignment as in MOOCs courses (Sandeen, 2013).

During this study, eALGO has shown its effectiveness of assigning scores comparing to teachers that are inherently inconsistent and can be influenced by many facts. A long term goal of automated scoring is to be able to generate an accurate formative feedback. This aspect would be studied in the future whether or not eALGO has the capability to give a formative feedback to students and how much this feedback can aid learners to progress and to develop their programming skills.

References


