Je-LKS
Journal of e-Learning and Knowledge Society
The Italian e-Learning Association Journal
PEER REVIEWED COMMUNICATIONS
Vol. 12, n.2, 2016
ISSN: 1826-6223 | eISSN: 1971-8829

AN AUTOMATED ASSESSMENT TOOL OF FLOWCHART PROGRAMS IN INTRODUCTORY PROGRAMMING COURSE USING GRAPH MATCHING

Rym Aiouni[1,2]
Anis Bey[1,2]
Tahar Bensebaa[1]

[1]Computer Science Department, Laboratory of Research in Computer Science (LRI), Badji Mokhtar-Annaba University, Annaba, Algeria
[2] Ecole Préparatoire en Sciences Economiques, Annaba, Algeria

Keywords: Automatic Assessment Tool, Learning and teaching Programming, Algorithms, Graph matching, Similarity.

Assessing students' programs by hand constitutes a burdensome task for assistant teachers in computer science course because the number of students and the variability of programs for one problem. So it will be important to support teachers and students in programming initiatives by considering new assessment approaches. In this paper, we propose an automated flowchart algorithms scoring system called eAlgo. Recognition of solutions is based on graph matching. Given parameters to the similarity measure, this method called AMAS (Automatic Matching for Algorithmic Solutions) will be able to assess automatically learners' algorithms based on a predefined algorithms pre-established by the instructor. The main goal of the proposed method is to help teachers to alleviate the scoring load.

for citations:
Aiouni R., Bey A., Bensebaa T. (2016), *An automated assessment tool of flowchart programs in Introductory Programming Course using graph matching,* Journal of e-Learning and Knowledge Society, v.12, n.2, 141-150. ISSN: 1826-6223, e-ISSN:1971-8829

1 Introduction

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.

Programming is a discipline used for a long time in a naïve 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 discipline taught. 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 favor 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 factorize in the algorithm, the set of behaviors of the task. The result is a "blank page syndrome", highlighted in particular by Kaasboll (*Ibidem*). 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 (Amerind *et al.*, 1998; Benabbou & Hanoune, 2007).

The TEL (Technology Enhanced Learning) 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 (Higginppàs *et al.*, 2005).

The main difficulty associated with this state of facts, just the assessment itself. Assessment in classroom was always the mysterious task (Hadji, 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 characterized 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, 2005). Localization 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 *et al.*, 2011).

In this paper, we have developed an automatic assessment system to assess students' algorithms for learning. AMAS is the proposed method to assess 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 as 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.

2 Learning by doing

Generally, people enjoy about learning new ways to better their selves. Learning programming is a discipline that requires this kind of learning approach (Caignaert, 1988) because students gain a better understanding of what it actually means to do the algorithm. Moreover, they get a deeper understanding of the programming task especially when they propose an erroneous solution and after that they obtain a formative feedback which promotes critical thinking skills (Labat, 2002). In this way, we have proposed a strategy to reuse common solutions as a basis and trying to find the most similar algorithm that match the proposed solution by learner with using s similarity measure between flowcharts. In the following section we explain the proposed method of assessment used the proposed tool.

3 eALGO, an automated assessment tool

3.1 Modeling an algorithmic solution

It is well known in programming that to simplify a complex task, we have to break it down into less complex tasks and repeat this process until we reach a level of decomposition with basic operations and / or elementary tasks. The algorithm solving the problem will then be a composition of these latter operations (basic and elementary). The number of decomposition stages depends on the complexity of the problem: more complex, 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 algorithm is at its last level when refining contains only basic operations, critical core operations, elementary operations and control structures.

> "*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.*" (Seymou Papert, in 'Mindstorms'[1])

A basic operation is defined as an operation known as algorithmic sorting a table. A basic operation is critical when its presence in the algorithm is essential. An elementary operation, meanwhile, is a simple algorithmic operation (e.g. assignment).

Thus, at level 1, the problem is decomposed into a set of basic operations (if detectable at that level), elementary operations and decomposable operations that can be linked by control structures. The number of levels of decomposition depends on the complexity of the problem. Down through the levels, only decomposable operations are broken, and this decomposition stops when you get to a level consisting only of basic operations and basic operations

This approach prevents the learner from drawing in the details from the start and gradually decreases the complexity of the problem being addressed. In addition, the learner can freely express his solution, without any influence or restriction, which promotes autonomy.

Our goal with this approach is to evaluate algorithmic solutions. However, it is important not to overlook the essential fallout that constitutes learning by learners of decomposition. Indeed, it is a must for the learner in the formulation

[1] Mindstorms: Children, Computers, and Powerful ideas (Basic Books (AZ))-Trade paperback (1993)

of the solution.

Unlike to the representation used in the work presented by Bey and Bensebaa (Bey & 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 analyzed in more effective way and thus makes program modification easier for learners.

Our goal is to arrive at a reliable estimate for algorithmic solutions. So when the learner has completed his decomposition of the problem that has been proposed, its solution is compared with those of the expert solutions together in a plan.

An algorithmic problem is characterized by its multitude of solutions and its many forms of decomposition. For this, we define plan solutions as a set of paths representing the different solutions that have the same decomposition approach to solving a problem. It can contain the correct steps as well as the wrong ones. It is made by an expert and has steps, correct or incorrect, considered pedagogically interesting (Figure 1).

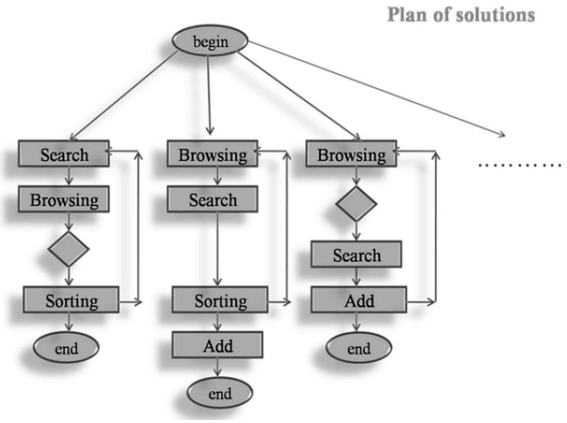


Fig. 1 - Example of a plan of solutions

3.2 How to assess the flowchart of the learner?

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 summarize the whole process according to three principal components as depicted in figure 2.

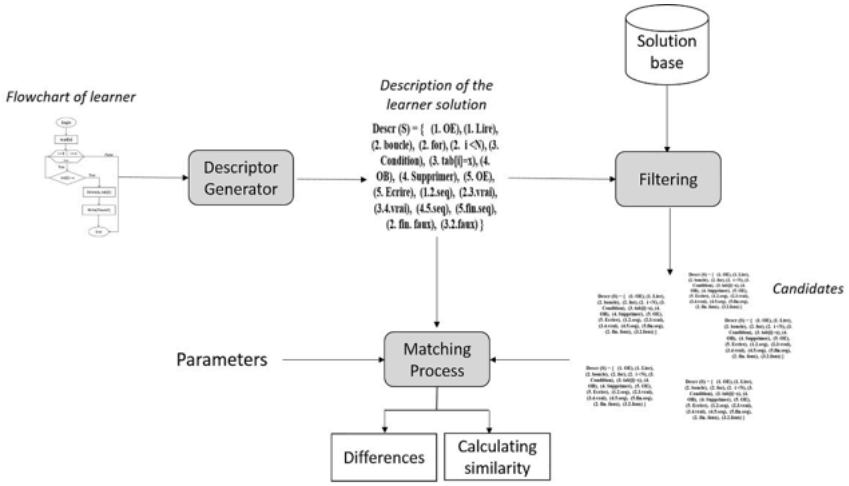


Fig. 2 - Functional architecture of the proposed method

3.3 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 (Sorlin *et al.*, 2006) on the measurement of multi-labeled graphs. This approach allowed us to propose a method for matching algorithmic solutions.

From the organizational 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 (*Num_operation*, *label*) and triples (*Num_operationS*, *Num_operationT*, *label*) are descriptors and constitute the description of the solution. The couple (*Num_operation*, *label*) describes operations of the flow chart and the triples (*Num_operationS*, *Num_operationT*, *label*) transition between operation (S and T are for source and target operation).

Given a set Lo of operation labels and a set Lt of edge labels, a flowchart algorithm is defined by a triple S = <O, ro, rt> such that:

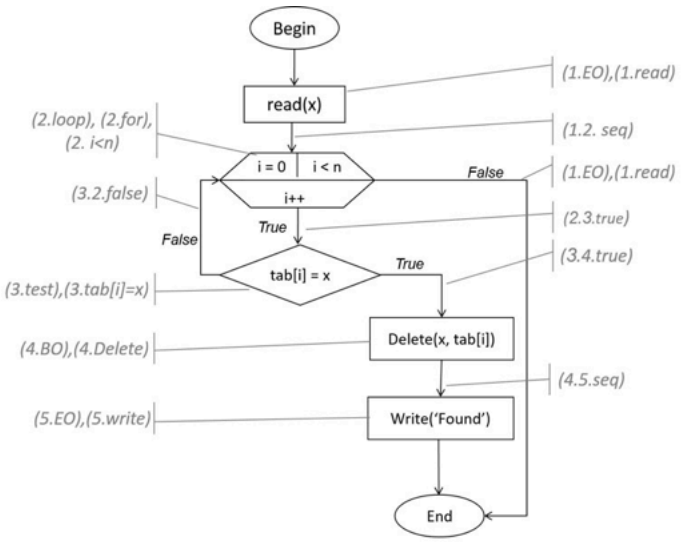
- O is a finite set of operations,
- ro $\subseteq$ O × Lo is a relation associating labels to operations (edges in the flowchart), i.e. ro is the set of couples (op_i, l) such that the operation

op_i is labeled 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 4, for the operation *read(x)* two descriptors were assigned; (*1.EO*) to mention the type of the operation and (*1.read*) for labeling.

- rt $\subseteq$ O × O × Lt is a relation associating labels to edges, i.e. rt is the set of triples (op_i, op_j, *l*) such that edge (op_i, op_j) is labeled by *l*. Label in transition may take two values: *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 Desc (S) = ro U rt.

As shown in Figure 3, the organizational labeling of a flowchart is as follows:



Descr(S)={*(1.EO),(1.read),(2.loop), (2.for),(2. i<n), (1.2. seq), (1.EO),(1.read),(3.2.false),(3.test), (3.tab[i]=x), (4.BO),(4.Delete), (5.EO),*(5.write),. (2.3. true), (3.4.true), (4.5.seq)}

Fig. 3 - Labelling process of the flowchart

3.4 Filtering

The filtering step is installed before the matching process in order to optimize the process of searching the most similar flowchart. It consists to find among the predefined flowcharts those who contain all the critical basic operations 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.

3.5 Matching process

Calculating the similarity between solutions

A match between two steps S1 = <O1, RO1, rt1> and S2 = <O2, RO2, rt2>, is a relationship: m $\subseteq$ O1× 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) generalized by Sorlin (Sorlin *et al.*, 2006):

$$Sim(S1,S2) = \frac{f(\text{descr}(S1) \cap \text{descr}(S2))}{f(\text{descr}(S1) \cup \text{descr}(S2))} \quad (1)$$

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:

$$f(F) = \sum_{(O,l)\in F} Weight(O,l) + \sum_{(O_1,O_2,l)\in S} Weight(O_1,O_2,l) \quad (2)$$

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 maximizes the overall similarity between flowchart algorithms. The best match is then scored according to the given

mark scheme provided by the expert.

The following figure demonstrates an example of matching two flowchart solution.

Descr (S_learner) =

{ **(1'. EO), (1'. read), (2'. loop), (2'. for),** (2'. i< N+1), **(3'. Condition), (3'. tab[i]=x), (4'. BO), (4'.Delete), (1.2.seq), (2.3.true), (3.4.true),** (4.fin. seq), **(2. end. false), (3.2.false)** }

Descr (S_base) =

{ **(1. EO), (1. read), (2. loop), (2. for),** (2. i< N), **(3. Condition), (3. tab[i]=x), (4. BO), (4. Delete),** (5. EO), (5. write), **(1.2.seq), (2.3.true), (3.4.true),** (4.5.seq), (5.end. seq), **(2. end. false), (3.2.false)** }

Matching M = {(1,1') , (2, 2') , (3 ,3 ') , (4, 4') }

$$Sim(S\_learner, S\_base) = \frac{f(\text{descr}(S\_learner) \cap \text{descr}(S\_base))}{f(\text{descr}(S\_learner) \cup \text{descr}(S\_base))} = \frac{4}{9} = 0.44$$

Fig. 4 - An illustrative example of similarity measure between learner solution (S_learner) and a solution from the base (S_base)

Conclusion and future works

In this research, we have proposed an automated assessment method of flowchart programs based on program matching offering to novice programmers an environment of practice and to have an instant correction of their solutions. The aim of this method is to let learners practicing programming without fear of error and to reduce the burden of grading students especially in the case of large number of assignment as in MOOCs courses (Cathy, 2013).

An experimental study in labs is in progress to measure the effectiveness of the system by comparing it against human experts' assessment results.

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

Amerein S. B., Proquin M., Renaud C. &Trigano P. (1998), *De la réciprocité éducative dans le cadre d'une nouvelle pédagogie de l'enseignement supérieur: un didacticiel au service de l'informatique fondamentale*. NTICF'98, ROUEN

Benabbou F., Hanoune M. (2007), *Utilisation des NTICs pour l'apprentissage et l'autoévaluation de l'algorithmique*. SETIT 2007.

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 (IJIL), 14(2), 2013.

Caignaert, C. (1988), *A study of learning methods evolution and programming progress*, Bulletin of EPI, 50, 52-60.

Cathy Sandeen (2013), *Assessment's Place in the New MOOC World*, Research and Practice in assessment, Vol. 8, 2013.

Charle, H. (1997), *L'évaluation démystifiée*. Paris, ESF, 1997.

Guibert N., Guittet L. & Girard, P. (2005), *Apprendre la programmation par l'exemple: méthode et système*, Proceedings de la 17eme conférence Francophone sur l'Interaction Homme-Machine 2005, Montpellier: 25-27 May, 461-466.

Hannola L., U. Nikula, M. Tuominen and H. Kälviäinen (2010), *The front end of innovation - a group method for the elicitation of software requirements*, International Journal of Innovation and Learning 7(3): 359-375.

Higgins C. A., Gray G., Symeonidis P. & Tsintsifas A. (2005), *Automated Assessment and Experiences of Teaching Programming*. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 5.

Kaasboll, J. (2002), *Learning Programming*, University of Oslo.

Labat, J. (2002), *EIAH: Quel retour d'information pour le tuteur ?*, Actes du colloque Technologies de l'information et de la Communication dans les enseignements d'ingénieurs et dans l'industrie Lyon.

Sorlin S., Sammound O., Solnon C., Jolion J.-M. (2006), *Mesurer la Similarité de Graphes*, Actes des 6e journées francophones Extraction et Gestion des Connaissances 2006, Atelier ECOI 2006, janvier 2006, p. 21-23.

Tversky.A (1977), *Features of similarity*, Psychological Review, 84, 327-352.

Wang T., Su X., Ma P., Wang Y., Wang K. (2011), *Ability-training-oriented Automated Assessment in Introductory Programming Course*, Computers&Education, Elsevier, vol. 56, pp. 220-226.