An e-Assessment Approach of Algorithmic Problem-Solving Skills

Anis Bey
Laboratory of Research in Computer Science -LRI-
University of Badji Mokhtar
BP 12, Annaba 23000, Algeria

Tahar Bensebaa
Laboratory of Research in Computer Science -LRI-
University of Badji Mokhtar
BP 12, Annaba 23000, Algeria

ABSTRACT
This work proposes a computer assisted assessment approach of algorithmic competencies. Aside from the fact that this approach brings a solution for delicate problem of e-assessment of algorithmic, in addition it is formative. Drawing one's inspiration from the basic principles of the algorithmic field itself, it recovers an interesting efficiency. It reclines on use of scalable solutions basis. Any learner's production, for a given problem, is automatically assessed if it is recognized, or assessed by a human expert if necessary. In the last case, it will enrich the basis if it is judged pedagogically interesting The purpose of this approach is to provide a formative and diagnostic assessment in order to empower the learner to acquire problem-solving skills of algorithmic.

General Terms
Technology Enhancing Learning, eAssessment, Approach of understanding program

Keywords
Algorithmic, problem-solving skills, e-assessment, competencies

1. INTRODUCTION
Algorithmic is a domain which fosters, especially, acquisition of a method of work and reflection, and development of analyze abilities, of abstraction, of anticipation and logic [1]. It is the kernel of any training of a computer scientist. It is also a material which has often been a source of problem for teachers and students. For teachers, because they have to find adequate methods to do assimilating abstract concepts to students who are only in their initiation phase. For students, algorithmic, unlikely to other sciences such as physic, does not offer for beginner an artless model viable of computer, which they can use it as a base to construct mental models more sophisticated. On the contrary, the experience student with this seems to favor an anthropomorphic modeling, which do not allow student to understand the brute error return which it confronted in the beginning of her/his algorithmic practice [2].

This finding is not recent. Since longtime, many experienced teachers in many universities have been in spite of their experience confronted to difficulties of their students face this material. Consequently, a very high dropout rate or failure is engendered during programming course, at the first cycle of university, which varies between 25 and 80% around the world as evidenced by Kaasboll in [3] and many students need to retake the course.

This situation drove several specialists to study and to strive to understand the reasons of this problem [4] [5] [6]. They show up that these difficulties are related principally to the nature of this field and situated mainly at the abstraction level of its concepts.

This abstraction prevents students to rely on real model to assimilate.

Developing solutions to algorithmic problems constitutes a major challenge for novices taking a CS1 course. Students have difficulties in formulating an idea for a solution, recognizing similarities among problems, and identifying familiar subtasks in a compound problem; consequently, they frequently end up with incorrect and cumbersome algorithms [7] [8] [9] [10]. Unfortunately, students are often overwhelmed by the many new ideas and details they need to comprehend in this short period of time.

At our university, the problem is more eloquent. The alarming failure rate (about 70%) of students in algorithmic material had finished by constituting a bottleneck in their progression at the 2nd to the 3rd year of license (the first cycle of university Computer Science CS). The study done by [11], initiated by the research group in TEL of Laboratory of Research in Computer Science, confirm all the inherent difficulties and that students have a difficulty to get around. For this reason, we recommended to address to Information and Communication Technology ICT in order to improve the training quality and to search how constraints of classroom formation can be creatively liberating and thus reduce the drop-out rate. In this sense, our research group advocated the development of a TEL environment of algorithmic able to supervise and to accompany disoriented students in algorithmic.

In this paper, we will discuss our research framework where we will introduce the difficulties of algorithmic and assessment in algorithmic. After that, in section 3, we propose our approach and the demarche followed for modelling and developing the assessment approach.

The fourth section presents the improvement of the approach. The paper is ended by a conclusion and a future work.

2. RESEARCH FRAMEWORK

The project where inscribes this work aspires the improvement of learning algorithmic at the university of Badji Mokhtar Annaba in Algeria. Based on learning by solving problem, this project aims to conceive a learning environment dedicated. It has as major goals giving responsibility to learners and bringing them to take aware of their insufficiencies in knowledge to understand, to practice for algorithmic thinking and to train for algorithmic problem solving skills.

In this project, we look after the assessment of competencies of algorithmic conception. For learner, this requirement to acquire problem solving skills is fundamental [12] [13]. The survey shows that many existent environments learning for algorithmic are focused on programming, i.e. how to write a set of instruction to resolve a problem, and that in the few works about algorithmic, assessment has been misconstrued and sometimes reduced to its simple expression (Multiple Choice Questions 'MCQ' …). Even if several methods and tools have been devoted to the assessment in TEL environment, they suffer insufficiency. This insufficiency is characterized either by inefficacity, doubtful result, or by uniqueness, i.e. they can't be applied into all fields (e.g. we can't assess algorithmic skills using MCQ). However, the positive impact of an appropriate assessment on learning is guaranteed [14] [15] [16]. With an appropriate assessment we mean an effective assessment which aims depth learning, detection of progress and learners gaps.

It's important to remember that assessment holds a preponderant place in large number of pedagogical activities [17]. Assessment in learning at classroom or in a Technology Enhanced Learning (TEL) environment was always source of ambiguity among evaluator and learner, and sometimes among evaluators themselves. Thus, assessment is rarely considered because it is often absent and obsolete [16]. Nevertheless, it is an integrate process in pedagogy. Furthermore, assessment is not restricted to attribute a mark, although this is important as far as we want to quantify skills. Evaluation constitutes a guide hall for learner's progression and intervenes in the interaction level between teacher and learner to optimize the transfer and the purchase of knowledge, skills and practices. It overtakes so theoretical framework. So, its importance is capital.

In addition, algorithmic is characterized by the multitude of solution for a given problem. This characteristic increases exponentially the assessment process in learning systems; it's a difficult task to expert of field to find all possible solution for a given problem in order to integrate them in the solution basis (indeed, expert human always forgets them).

Through this ensemble of obstacles that we can measure the size of difficulties toward pass round to strive for automatic assessment of algorithmic competencies.

3. OUR PROPOSITION

It is known in algorithmic that to execute complex tasks, every task must be decomposed on succession of simplest tasks. This decomposition is repeated until having elementary tasks. The number of step of decomposition depends on the complexity of problem to resolve.

This descendant approach (also named divide-and-conquer), source of our inspiration, allows to pass gradually and with a maximum chance to success, from the abstract description of the problem solution (with a complex process) to an algorithm resolving the problem [18] [19]. We can say that an algorithm is on the last level of decomposition when it contains only elementary operations, basic operations (known operations) and control structures.

We define a basic operation (BO) such as an operation well-known in algorithmic like Sorting, Researching, etc. whereas elementary operation (EO) is a simple instruction such as assignment.

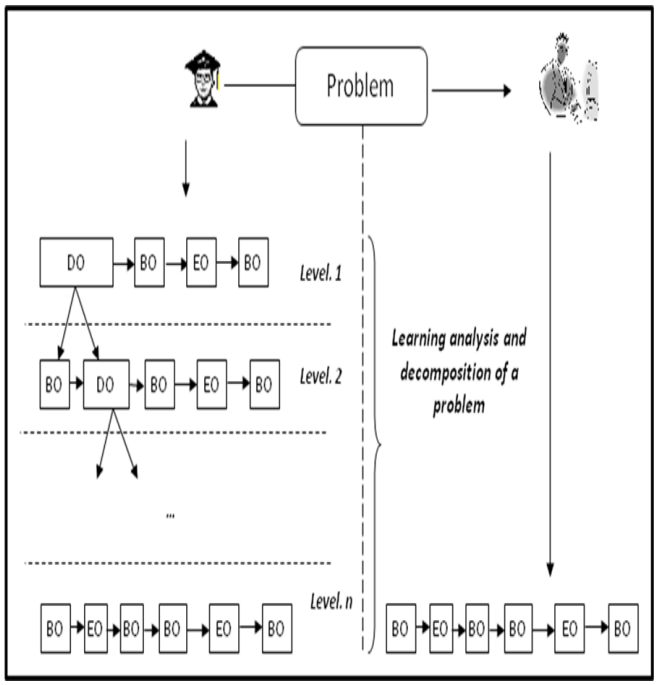

Fig 1: Decomposition of a problem

Figure 1 illustrates that at level one, the problem is divided into an ensemble of decomposable operation (DO), basic operations and elementary operations connected by control structures if necessary (condition, repetition,…).

At level two, every decomposable operation of level one is decomposed, either at decomposable operation again, or at basic operations or elementary operations. This depends on the complexity of the problem.

Etc.

Thus, in an assessment context, learners may present their solutions, to a given problem, in terms of decomposable, basic and elementary operations. These operations may be connected by control structures (loops, condition). This gradual decomposition of solution and the use of high-level concept (basic operations) allow firstly an efficient assessment, and next totally adapted to the field.

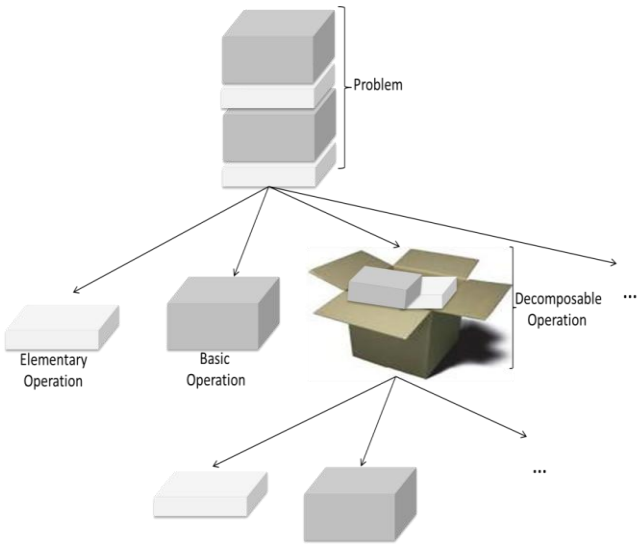

Fig 2: Decomposition approach

This manner to do aims two objectives:

- To compel learner learning to divide a problem. This teaches him/her to decrease gradually the complexity of problem and evades him/her drowning in details at first,
- To let learner focus its efforts on the problem and not on secondary questions (basis operations). For example, in the sorting problem, what interest us is if learner opts for a sorting and not how he/she makes sorting. However, basic operations may do themselves a learning object.

3.1 Modeling and assessment

The variety of solutions for an algorithmic problem makes situation where learner can take a 'way' among several which represents a solution of the problem (see Figure 3).

We call a solution plan, a path constituted with basic and elementary operations. This plan describes a correct solution for a given problem, as may describe an erroneous solution pedagogically interesting. The whole of plans (SP) constitutes a Descriptive Map (DM) of the problem (see Figure 3).

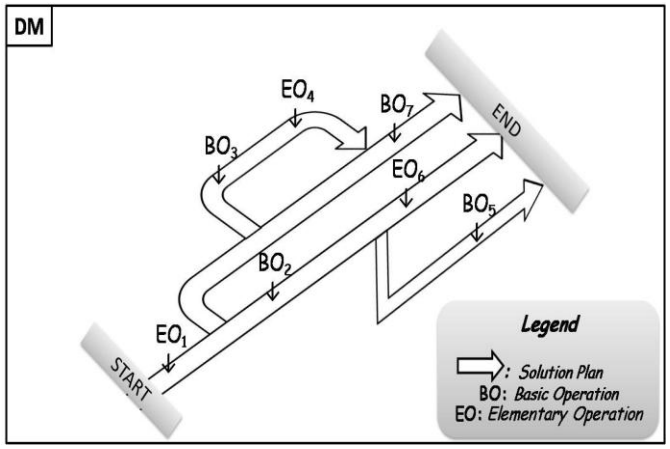

Fig 3: Descriptive Map

This descriptive map allows to locate learner and to recognize its step (right or erroneous) during solving problem.

Holding BO (organizing in a library) and EO, learner can express his/her solution freely without any restriction neither influences.

Teacher, for its part, should define beforehand for every problem a DM. We know in advance that it will not be complete (describes all solutions/errors of problems), but he/she must foresee the most plausible SP (SP having more probability to appear), leaning on his/her experience and given for each SP an interpretation. During the definition of a SP, teacher attributes for each BO/EO a weight expressing its importance in the solution. The assignment of weight depends on the pedagogical objectives aimed by the problem.

DMs are scalable. Overtime, they can be enriching by other SP, i.e. integrating other solutions proposed by learners but not foresee by teacher. Indeed, every solution not recognized, its evaluation should be suspended until the intervention of a human teacher which if he/she judges it interesting (pedagogically interesting), will add it into the DM.

3.2 Assessment script

The produced assessment is an individual, summative, diagnostic and formative assessment. To assess a plan proposed by learner, it must recognize in the DM the nearest SP (called the referent SP). This one will serve to calculate the mark. The recognition of referent SP is done by the formula (1) which calculates the similarity degree of the learner plan with all SP of DM of the problem. This formula is conceived on the basis of two parameters, succession of BO/EO (the order) and use of an inappropriate BO/EO [18] [19] [20].

$$SD = \frac{Nb'S}{Nb} - \left(P_1.10^{-1}\frac{Nb'SD}{Nb} + P_2.10^{-1}\frac{Nb'I}{Nb} \right) \quad (1)$$

- (Nb'S/Nb): Represents the similarity operation by operation,
- (Nb'SD/Nb): Represents the penalty for BO/EO which are disordered,
- (Nb'I/Nb): Represents the penalty for BO/EO which are useless,

Where:

- Nb: number of BO/EO of the DMs' plan,
- Nb'S: number of BO/EO of the proposed plan, which are similar to BO/EO of DMs' SP,
- Nb'SD: number of BO/EO of the proposed plan which are disordered,
- Nb'I: number of BO/EO of the proposed plan which are useless.

4. EVOLUTION OF THE APPROACH

The experimentation of this approach through the realized prototype [20] revealed an important heaviness (compound!!) in looking for the nearest solution. In fact, two solutions including

identical and independent operations in a different order constitute two different solutions. This has as result increasing exponentially the size of the DM and consequently the basis of plans.

We redefined then the concept of the plan. And, instead of considering a plan like a suite of operations (basic and elementary), it becomes a canonical formalism for the representation of a task. This redefinition will allow a better comprehension of learner's solutions because the explanation of their objectives and the identification of their structures, conceptions, and execution behaviors will be well translated. This is well illustrated with the following section.

5. AN APPROACH OF UNDERSTANDING PROGRAMS BASED PLAN FOR AUTOMATIC ASSESSMENT

Program understanding has been frequently defined as a recognition process of program plans in a fragment of source code [21] [22] [23] [24] [25].

Research works on program understanding have as goals to identify conceptual information and to develop concepts and extraction tools from existent and operational systems. Two types of extraction concepts have been identified: program plans and program slice. The first linked to tasks implemented by a program, the second concerns the execution behavior of a program.

Our aim is the definitional aspect of a solution and the manner for solving problem. The comprehension oriented task constitutes an interesting approach for our problem. This approach requires knowledge basis containing a set of standard forms (tasks description). These standards forms are represented under canonical form using the formalism of plan. A program plan corresponds to a fragment of code which achieves a stereotyped action.

In program diagnostic approaches based on the concept of plan, a program is seen like a set of tasks to make [26]. The set of implicit tasks in the program constitutes the conception of the last one. Semantically, a program contains implicitly a high level of abstract concepts and contains explicitly language concepts.

Indeed, every task corresponds to an abstract concept. We call a program plan a description of correspondences between a task and its sequence of instructions [22]. In other words, a program plan is a recognition rule of an abstract concept from language concepts specified in source code.

A plan is defined as following:

```
Plan    c (list of attributs)
Consist of      cc1, cc2... ccn
Such that       e1, e2... er
```

Where:

- **C:** represents the name the plan with a list of attributes;
- **Consist of:** regroups components of the plan;
- **Such that:** constraints representing the different dependences between the different components of the plan,

In our case, the plan becomes a description of correspondence between a task (problem to resolve) and its operation sequence instead a sequence of instructions, i.e. instead using only instructions like a component in the plan, it is possible to use also a high level of operations. Thus, the plan components (cc1, cc2… ccn) represent either basic operation BO which are high level concepts, or elementary operations EO, or also other tasks. The following example illustrates a plan of a counter.

```
Plan     counter     (c:?c,     test:?test,
body:?body);
/*c, test and body represent attributes of
the plan*/
Consist of
EO: Init-counter: zero (var:?c);
/* initialization of the counter */
EO: Inc-counter: increment (var:?c);
/* to increment the counter */
BO Repeat:  loop (test:?test, body: ?body);
/*loop repetition*/
Such that
Init-com:    Control-flow    (Init-counter,
Repeat);
Flow1-of-counter:   Data-dep   (Inc-counter,
Init-counter,?c);
Flow2-of-counter:  Data-dep  (repeat,  inc-
counter,?c);
```

The dependences considered in constructing of the program plans are data flow and control dependencies connecting concepts between them to form a high level concepts. These relations are described as predefined functions can be assessed true or false.

Our aim is not assessing language concepts (the set of instructions), but assessing the high level abstract concepts which are used to resolve a problem, i.e. to externalize the abstract concepts used (Research, Deleting, etc.) as plan components and their organizations (operation chaining and constraints) to assess algorithmic problem solving skills of learner.

5.1 A new solution basis

These new solutions (using the formalism of plan) constitute now the solutions basis. It describes tasks which resolve problems in term of basic and elementary operations or also using another task (see Figure 4).

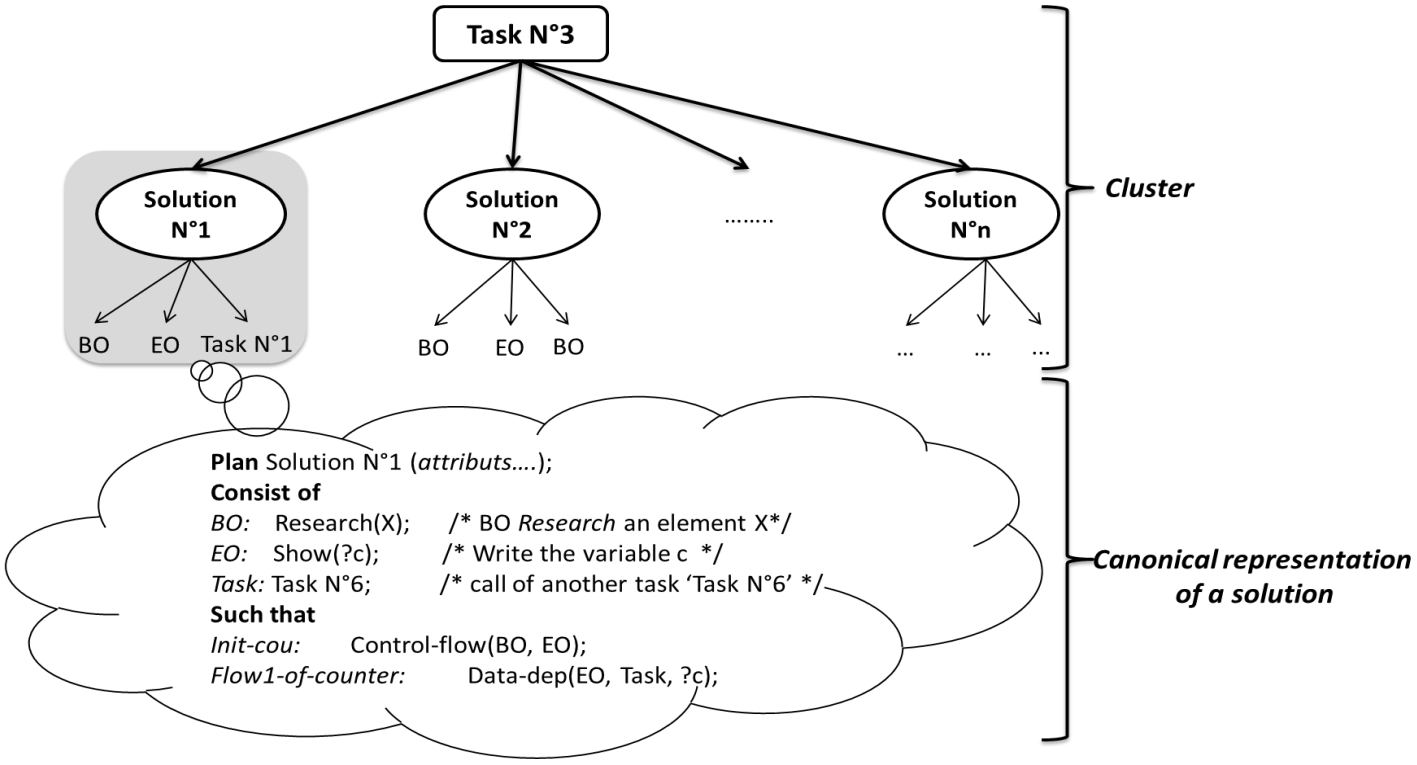

Fig 4: Solutions cluster with the canonical representation of the solution in the basis

5.2 Library of basic operations

The basic operations used in the conception of a solution, are regrouped in a library. A basic operation has parameters which are used in the unification during the plan recognition in the assessment phase. The structure of a basic operation is defined as following:

Name-operation (list of parameters);

5.3 Looking for the referent plan to assess

The figure 5 resumes the assessment script. Firstly, learner conceives a solution plan using three types of bloc: BO bloc, EO bloc or control structure bloc. The syntax used to define a bloc is a pseudo code. After that, this solution is translated to a plan separating operations from their constraints.

The research of the most similar plan to the learner's solution in the solution basis passes by measuring the similarity between the learner's plan and the basis's plans using Jaccard similarity coefficient [27]. The Jaccard coefficient measures similarity between sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets (2). Two sets A and B, the coefficient is:

$$S = J(A,B) = \frac{A \cap B}{A \cup B} \qquad (2)$$

To compare two plans, we must measure the similarity coefficient between components and constraints of the first plan like one set (A), and components and constraints of the second plan like another set (B). The comparison is done by comparing the learner's plan with all plans of the treated problem and which have higher similarity index is a candidate to be the referent plan.

The referent plan is selected only if the similarity index is superior to the threshold fixed by teacher. In this case, the solution proposed by learner is recognized. And in order to diagnosis the learner's solution, we have to measure two similarity (using Jaccard coefficient). On the one hand, the first similarity S1 is between the components of learner's plan and components of the referent plan, and on the other hand, the second similarity S2 is between the constraints of each one. This matching between components and constraints allow us to know if learner has made the error at the level of choosing the appropriate operation to resolve the problem or at the level of control structures to link operations.

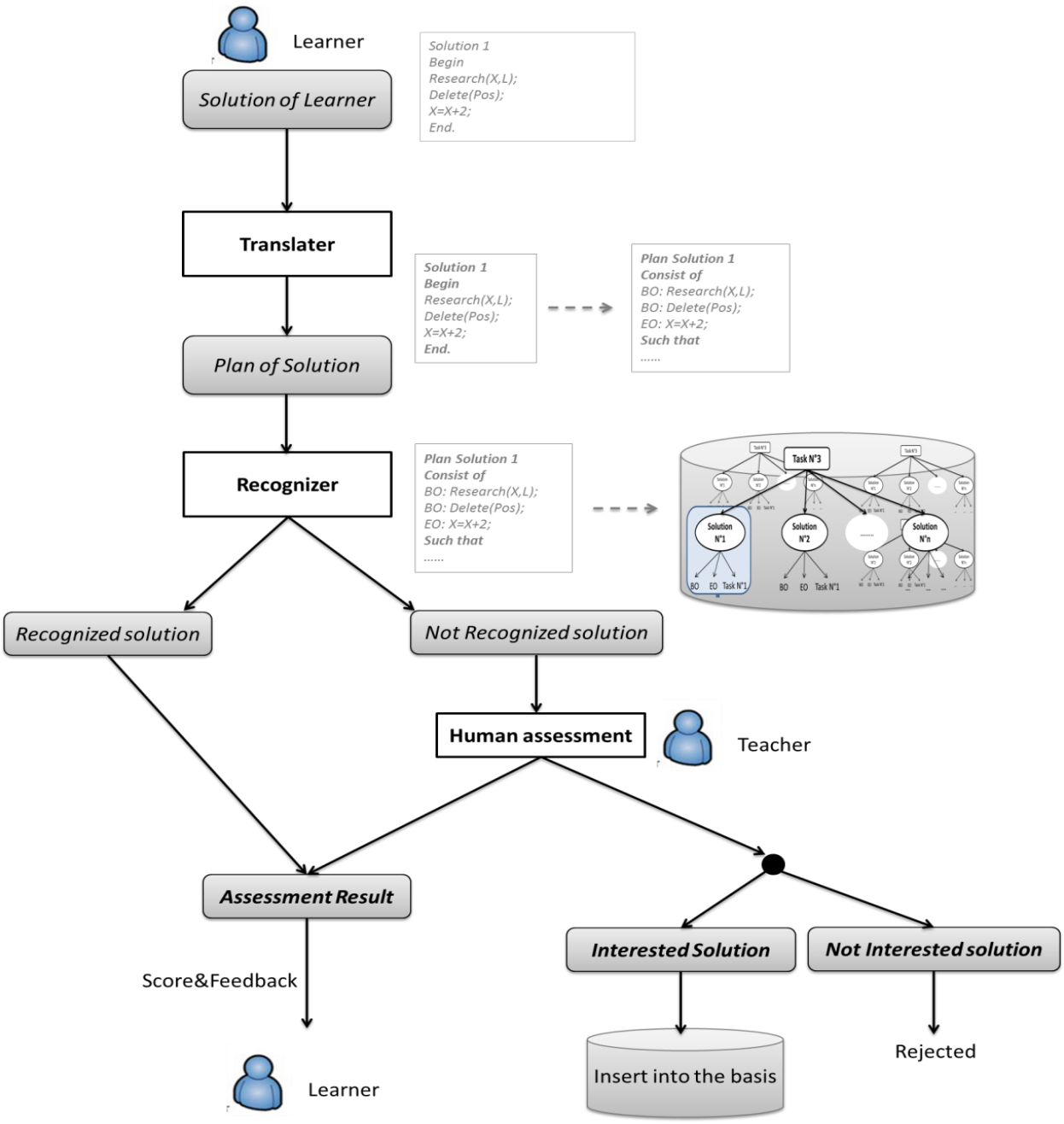

Fig 5: Assessment Script

Using this sub-matching after looking for the referent plan helps to diagnosis gaps of learners and to differentiate between the skills of solving problem and the knowledge used to resolve a problem.

6. CONCLUSION AND FUTUR WORK

The proposed approach is an assessment approach of algorithmic competencies and specially knowledge which intervenes in solving problems. The provided assessment is formative (to aid learner to progress) and summative (to quantify competences and knowledge really owned and to attribute a mark) assessment.

Each proposed demarche is assessed automatically or by a human expert (Deferred assessment). In the last case, the solution proposed by learner may be enrich (if expert judges it interesting) the solution basis.

Even if it has been conceived for algorithmic competencies assessment, this approach can easily be adapted in any field which manipulates procedural knowledge (know how).

7. REFERENCES

[1] Müldner T., Shakshuki E. 2003. Teaching Students to Implement Algorithms. Jodrey School of Computer Science, Acadia University. TR-2003-03.

[2] Caignaert C. 1988. Etude de l'évolution des méthodes d'apprentissage et de programmation. Le bulletin de l'EPI N°50.

[3] Kaasboll J. 2002. Learning Programming. University of Oslo.

[4] Soloway E., Bonar J., Ehrlich K. 1983. Cognitive strategies and looping constructs: an empirical study. Communications of the ACM, Vol. 26, n° 11, p. 853–860.

[5] McCracken M., Almastrum V., Diaz D., Guzdial M., Hagan D., Kolikant D., Laver C., Thomas L., Utting I., Wilusz T. 2001. "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students". ACM Sigcse Bullettin, Vol. 33, n°4, p. 125-140.

[6] Lister R., Adams E., Fitzgerald S., Fone W., Hamer J., Lindholm M., et al. 2008. A multinational study of reading and tracing skills in novice programmers. Working group reports from ITiCSE'04 on Innovation and technology in computer science education. ACM Digital Library, p.119–150.

[7] Deek, F.P., & McHugh, J. 2000. "Problem-solving methodologies and the development of critical thinking skills". Journal of Computer Science Education, 14(1-2), 6-12.

[8] Du Boulay, B. 1986. "Some Difficulties of Learning to Program". J. of Educational Computing Research, 2(1), 57-73.

[9] Joni, S.A. & Soloway, E. 1986. "But my program run! Discourse rules for novice programmers", J. Educational Computing Research, Vol. 2(1), 95-126.

[10] Robins, A., Rountree, J., & Rountree, N. 2003. "Learning and teaching programming: a review and discussion". Computer Science Education, 13(2), 137-172.

[11] Bensalem H., Bensebaa T. 2010. Contribution to the improvement of learning algorithmic. 10th International Educational Technology Conference (IETC) 2010, Istanbul, Turkey, April 26-28, 2010.

[12] Deek F. P., McHugh J. A. 1998. "A survey and critical analysis of tools for learning programming". Computer Science Education, Vol.8, n°2, p. 130-178.

[13] Ragonis N., Ben-Ari M. 2005. "A long-term investigation of the comprehension of OOP concepts by novices". Computer Science Education, Vol. 15, n°3, p. 203-221.

[14] Lasnier F. 2000. Réussir la formation par compétences. Montréal, Guérin.

[15] Bull J. 1999. "Computer-Assisted Assessment: Impact on Higher Education Institutions?" Journal of Educational Technology and Society, Vol. 3, n°2, p.123-126.

[16] Durand G., Martel C. 2006. Vers une scénarisation de l'évaluation en EIAH. 1ères Rencontres Jeunes Chercheurs sur les Environnements Informatiques pour l'Apprentissage Humain, RJCEIAH.

[17] Lorrie A. Shepard. 2000. The Role of Classroom Assessment in Teaching and Learning. CSE Technical Report---517 CRESST/University of Colorado at Boulder.

[18] Bey A., Bensebaa T., Bensalem H. 2010. Assessment of algorithmic skills in learning environment », The 2nd International Conference on Education Technology and Computer, Shanghai, China 22-24 June 2010.

[19] Bey A., Bensebaa T., Bensalem H. 2010. Assessment of algorithmic competences in Tel environment, 10th International Educational Technology Conference (IETC), Istanbul, Turkey, April 26-28, 2010.

[20] Bey A., Bensebaa T. 2011. Algo+, an assessment tool for algorithmic competencies. IEEE Engineering Education 2011 Learning Environments and Ecosystems in Engineering Education, EDUCON 2011, 04-05 April, Amman, Jordan. ISBN: 978-1-61284-641-5, 2011.

[21] Woods S., Yang Q. 1995. Program understanding as constraint satisfaction. Proceedings of the IEEE Seventh International Workshop on Computer-Aided Software Engineering, Toronto, Ontario, Canada, p. 318–327.

[22] Quilici A. 1994. A memory-based approach to recognizing programming plans. Communications of the ACM, Vol. 37, p. 84–93.

[23] Kozaczynski W., Ning, J. 1994. Automated program understanding by concept recognition, Automated Software Engineering, Vol. 1, p. 61-78.

[24] Wills L. M. 1990. "Automated program recognition: a feasibility demonstration". Artificial Intelligence, Vol. 45, p. 113–172.

[25] Johnson W.L. 1986. Intention Based Diagnosis of Novice Programming Errors. Los Altos, CA: Morgan Kaufman.

[26] Soloway E. 1984. A Cognitively-Based Methodology for Designing Languages/ Environments/Methodologies. Sigplan Notices - SIGPLAN, vol. 19, n° 5, p. 193-196.

[27] Jaccard P. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura, Bulletin de la Société Vaudoise des Sciences Naturelles 37: 547–579.