International Journal of Electrical and Computer Engineering (IJECE)
Vol. 11, No. 2, April 2021, pp. 1578~1590
ISSN: 2088-8708, DOI: 10.11591/ijece.v11i2.pp1578-1590

New method for summative evaluation of UML class diagrams based on graph similarities

Outair Anas, Tanana Mariam, Lyhyaoui Abdelouahid
National School of Applied Science, Innovative Technology Laboratory, Morocco

Article Info

Article history:

Received Feb 20, 2020
Revised Aug 26, 2020
Accepted Oct 16, 2020

Keywords:

Graphe matching
Learner assessment
Similarity measure
UML class diagram

ABSTRACT

This paper deals with the problem of the evaluation of the student's productions during the construction of a UML class diagram from textual speciations, which can be a tedious task for teachers. The main objective is to propose a method of summative and semi-automatic evaluation of the class diagrams produced by the students, in order to provide an educational reaction on the learning process, and to reduce the evaluation work for the teachers. To achieve this objective, we must analyze these productions and study the transformation, matching, similarity measurement and comparison of several UML graphs. From this study, we adopted a method based on the comparison and matching of the components of several UML diagrams. This proposal is applied to evaluate UML class diagrams and focuses on the structural and semantic aspects of the UML graph produced by students compared to several solutions proposed by the teacher.

Corresponding Author:

Outair Anas
Innovative Technology Laboratory
National School of Applied Science
E.N.S.A Route Achakar, BP 1818 Principal Tangier, Morocco
Email: anas.outair@gmail.com

1. INTRODUCTION

The evaluation of learners occupies a very important place in teaching. The knowledge acquired by the students can be tested by the teacher in the form form of a summative/certification evaluation, if the objective is to validate for example a unit of value, a course, a year or a diploma [1]. Indeed, evaluation is the process by which people make value judgments on a particular subject. In the learning process, this operation being already complicated at the base, takes on even more oversized proportions. In a teaching and learning community, the most effective assessment is one that encourages and rewards effective teaching practices based on learning outcomes [2, 3].

The assessment of learning allows the learner to identify his own strengths and weaknesses, and to determine the types of information he needs, to essentially correct his shortcomings [4]. When this assessment is used correctly, students learn that it is possible to start a self-assessment, in order to improve their performance throughout their lives [5]. In all existing education systems, assessment remains the only educational tool, which validates the achievements of students in order to access the following learning subject [6]. Although the evaluation process is very complicated at the outset, this operation becomes even more tedious for the teacher when it comes to evaluating the learner's know-how in complex systems [7]. The difficulty of this task increases further when the number of students increases, which is always the case in higher education.

In this context, this article is a contribution to research efforts on improving the evaluation process for both the teacher and the students. The problem posed is how can we facilitate the task of correction

related to learning complicated subjects, for example, UML diagrams. Thus we have made a semi-automatic evaluation system for comparing the diagrams generated by the students with any diagrams from the teacher. We have reformulated this problem by setting the following scientific objectives:

- Represent class diagrams in metamodel
- Propose a new formalism through the improvement of the metamodel
- Add new elements to the metamodel in order to be able to use all the properties of the case studies
- Identify any similarities between UML diagrams and graphs
- Develop a new similarity calculation method to evaluate the graphs.

2. RELATED WORK

2.1. Evaluation of student productions in general

There are several approaches and literature research that work in the area of evaluation. The assessment of artificial intelligence learners describes the design of an open learning environment designed to monitor students' understanding, evaluate their prior knowledge, build individual learner profiles, provide personalized assistance and finally evaluate their performance [8]. Janicic and Maria's research (2014) presents two methods that can be used to improve the automated evaluation of C language programs produced by students [9]. They are based on the software verification and the measure of similarity between the students' productions and the teacher's solution (s). Both techniques can be used to provide useful feedback to students and to improve automated rating for teachers [10].

Tanana's research (2009) sought to propose a formative evaluation method of the learner's knowledge based on the use of supervised classification algorithms. They have chosen digital electronics as their field of application. This method was intended to facilitate the assessment of learners to the teacher. This is more a "help with correction" than an automatic evaluation [11].

2.2. Environments used for teaching and evaluating UML language learning

Several object-oriented modeling courses adopt UML to teach analysis and design techniques. It is recognized that appropriate UML modeling tools should be used in conjunction with the taught subject in order for students to gain practical experience [12]. UML professional tools tend to be too complex and lack educational functionality [13].

Some UML language environments have been developed for professional use by experienced people. They are not suitable for pedagogical use and have many functionalities that could increase and brake learning in the learner [14]. Other environments can be used for learner teaching, but do not have assessment tools. Subsquently, we provide examples of UML language teaching environments focusing on their purpose, operation, advantages and disadvantages.

2.2.1. StudentUML

StudentUML is a simple but efficient educational tool that supports the construction of consistent UML diagrams. The goal of StudentUML is to provide students with a tool that meets their learning needs without diverting them from the learning process [15]. The most important educational feature of StudentUML is its ability to check the consistency of diagrams. Students construct diagrams that could be correct when they are examined independently, but erroneous when compared to other diagrams in the same project. These consistency errors do not allow students to correctly implement their models. StudentUML provides avility to automatically check the consistency between existing diagrams [16]. StudentUML is an open learning environment allows for the construction of UML diagrams, validate them and check their consistency, but it does not specify the semantic difference to other diagrams, and does not provide options for automatic correction [17].

2.2.2. KERMIT

KERMIT is an intelligent, knowledge-based entity-relationship modeling environment, designed for university students who are learning conceptual database modeling. The system presents the requirements for a database for which the student must design an entity-relationship (ER) diagram [18]. KERMIT is based on constraint-based modelling (CBM), a student modeling approach proposed by Ohlsson. This is a very effective approach that focuses on the key to individualized knowledge-based education. KERMIT is an open learning environment for database modeling, would prove very useful for practice. Moreover, the semantically rich feedback generated by the system and its ability to refine an individual student makes it an invaluable resource for students [19].

KERMIT is an open learning environment for database modeling that provides individual monotoring in the form of educational feedback to learners during the modeling activity [20]. Each feedback

is built here from the violation of a constraint (a mistake made by the learner). This environment has the advantage of allowing the text of the problem to be manipulated throughout the activity, but it forces the the elements to be edited against the statement. Indeed, the learner has no real opportunity to represent elements not explicitly specified in the statement. The teacher can add new exercises by defining the statement and an ideal solution corresponding in a dedicated teacher interface. The use of the environment is restricted to novices, and the authors advocate not to introduce implicit elements into the statement and to adapt the statements to contain as little ambiguity as possible.

2.2.3. Diagram

The diagram environment is designed to lead the learner, through interaction, to mobilize the three functions of metacognitive regulation and thus to facilitate the acquisition of the concepts of object-oriented modeling by generating the emergence of instrumented action schemes to perform effectively the prescribed task [21]. The Diagram environment includes a subset of the features of the traditional UML editors. It provides only the graphical elements needed to build an UML class diagram and simplifies editing of the different elements characteristics. In addition, Diagram provides the opportunity to work simultaneously with the statement (describing the specifications of the exercise to be modeled) and with the UML class diagram, which facilitates visual control of the modeling. This feature provides greater opportunities for interaction because the learner can select elements of the statement and change its visual aspect [22].

Diagram offers three types of modeling scenarios: The first is to build a complete diagram from a statement (this is the activity that is of particular interested to us). The other two scenarios consist of completing a partial diagram and correcting an erroneous diagram. This environment does not correct the learner‘s errors and is not intended to replace the teacher during the UML diagram construction. A diagram assists the learner in his work by encouraging self-correction. The teacher remains present during the modeling activity (conducted in practical work sessions) to provide advice to the learner.

2.3. Graph transformation of UML diagram

Otherwise, different approaches to graphic transformation can be found in the literature. We have studied the existing approaches relating to the transformation of a class diagram into a graph [23]. The transformation of graphs can easily model the graphical structure. It has become a modeling tool often used in the case of complex systems like the class diagram. The example below represents a transformation of a class diagram into a directed and labeled graph where the edges are oriented and multiple between the vertices which are either classes or attributes. The vertices and the edges have many characteristics. The advantage of this representation is to consider a class diagram in its simplest expression.

The representation in the form of a metamodel [24], as that defined by Holcher's studies very precisely describes all the elements of class diagram and the semantics of these elements. It also allows to clearly exposing its structure. For example, the Figure 1 shows an UML metamodel, the classes, attributes, operations and association ends are more specialized named elements. A class can contain attributes and operations which themselves can contain types. It has an association end that defines the role of the linked class as well as a multiplicity. An association can have two association ends. The advantage of this metamodel is that it is adapted to the OMG standard. For the disadvantage, necessary elements are not presented such as visibility, association class, type of association.

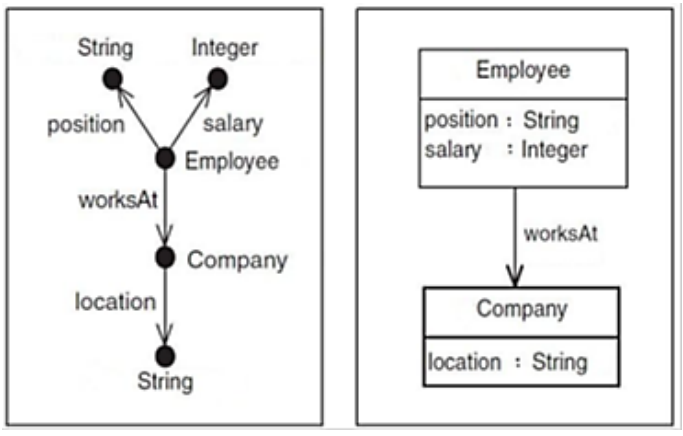


Figure 1. An attributed graph in two different notations

Based on the extract from the UML metamodel [25], we can transform a class diagram into a graph as shown in Figure 2. A class is a vertex has an edge towards the attribute, which is also a vertex and which can be typed. A class also has an association end, it is a vertex which contains several labels such as the type of relation and the multiplicity. This relation is named, it is linked by an aggregation with the other class. The inheritance relationship is represented by a labeled edge. This representation clearly expresses the links maintained in terms of their elements and their characteristics. They are made explicit using vertices and edges [26].

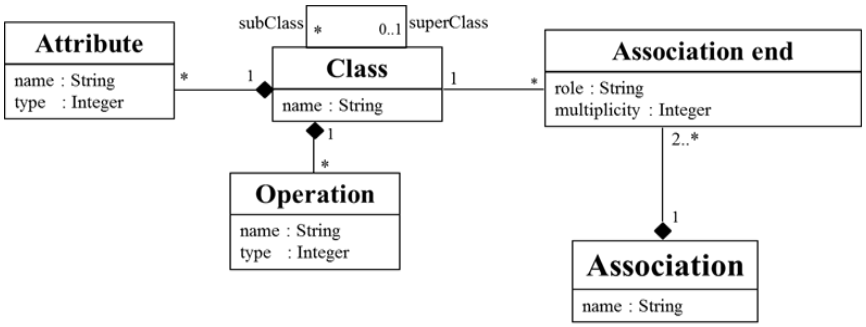


Figure 2. Improving metamodel class diagram

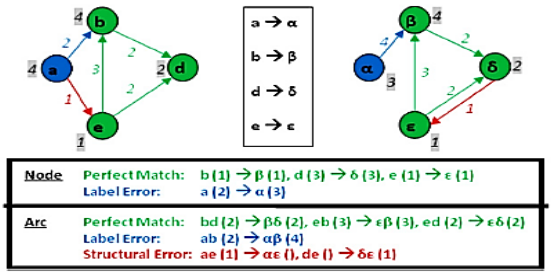
3. MEASURING SIMILARITY AND MATCHING UML GRAPHS

Our domain of application is the UML class diagrams. We will define a similarity measure between class diagrams transformed into a UML graph. We saw in the previous section that a class diagram can be represented by a UML graph. Our main objective is to compare the class diagrams produced by the students which are transformed into a UML graph with the diagrams of the teacher. For this we wish to define a similarity function which must be able to produce the correspondence, the difference and the detection of errors between these graphs [27]. To meet these different objectives, we studied the comparison of graphs using graph matching techniques and measures of node similarity. We will therefore build on our existing work on graph similarity measures to build our own method.

3.1. Matching approaches to graphs

Different matching approaches have been defined and applied as graph isomorphism [28] which allows to check if two graphs are structurally identical. The subgraphs [29] which allows you to check if a graph is included in another graph. The search for a larger common subgraph [30] and the calculation of the graph editing distance [31]. The problem of these matching was considered a complete NP and difficult NP problem. With the exception of graph isomorphism, complexity is not clearly defined. We have studied another technique, which consists in implementing a similarity measure and looking for matching [32].

We focused on vertice and edge level approaches [33]. The comparison of several elements of the graphs is based in particular on the evaluation of their similarity or their differences, then it consists in identifying and qualifying their common points. This study proposes a comparison of two graphs, for each vertex and edge of a graph are paired with several vertices and egdes of the other graph. The matching of the vertices will be defined thanks to the calculation of the similarity measure. The couples that have maximum similarity will be selected and stored in a correspondence matrix as shown in Figure 3.



$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} \alpha & \beta & \delta & \epsilon \end{array} \\ \begin{pmatrix} 0.88 & 0.22 & 0.55 & 0.35 \\ 0.19 & 1.00 & 0.65 & 0.43 \\ 0.37 & 0.57 & 1.00 & 0.23 \\ 0.20 & 0.33 & 0.19 & 0.8 \end{pmatrix} \end{array}$$

Figure 3. Example of a graph matching

We have defined our method as a matching system that follows three sequential steps. The first is a step of preprocessing the input diagrams, each class diagram will be transformed into a graph. The second is the matching process, it allows you to calculate similarities to each pair of elements. And the third returns as a formative evaluation of their paired elements with a list of differences and errors, and a summative evaluation to classify the compared diagrams.

3.2. Similarity measure

The comparison of two graphs is the task of identifying the semantic correspondences between the elements of two graphs [34]. This correspondence can be quantified in terms of similarity scores, which indicate the proximity of the two graphs. Therefore, their similarities and differences must be precisely quantified to have an exact match. The task takes time because comparing two graphs to assess their similarity is a kind of combinatorial problem generally called graph matching problem [35]. Therefore, an efficient comparison algorithm is necessary to avoid the complexity of the method and to provide an acceptable solution. Indeed, we improve this comparison by introducing one more metric and by revisiting the definitions of existing ones. Each time the couple is compared, a similarity measure is calculated, and is stored in the similarity matrix [36]. Finally, a mapping is determined and extracts the correspondences and the differences resulting from the comparison of the two diagrams as well as the proposal of the corrections of the errors committed by the students.

The properties which are relevant for the similarity of two nodes of the same type are either given by their attributes (for example the names), or by other nodes in the neighborhood of these nodes. We use a set of comparison functions to determine the similarity between two nodes. These functions compare two properties of the same type belonging to different nodes. They return a value between 0 and 1, a value of 0 means no similarity between the nodes, a value of 1 expresses equality [37].

Obviously, some properties are more relevant to the similarity of nodes than others. Therefore, weights and thresholds, which are external resources used by the matching process, must be assigned to each property. They are all configurable and can be adapted as required. The weights and thresholds should be chosen based on the semantics of the UML graph type and based on what users see as a significant change.

For each specific type of UML graph, a configuration file describes the similarity properties relevant of UML graph elements. Two elements of the same type are compared using a comparison function which returns a value between 0 and 1. The comparison function can be defined criteria for each type of element. The criteria take into account some parts of the elements depending on the types, and the actual structure of the compared UML graphs. The values of the different criteria are weighted, and the similarity value is calculated by addition, as can be seen in the following formula [38]:

$$Sim\ \ (e_1, e_2) = \sum_{c \in C} s_c \times compare_c(e_1, e_2)$$

where :
e_1 and e_2 are the elements to compare;
C is the set of criteria;
s_c is the threshold for criteria c;
$compare_c$ is the comparison function for criteria c.

The total similarity of two elements is assessed according to the elements they contain. If the elements admit relationships with each other, the evaluation of their similarities can be taken into account during calculation. The weight values are assigned and weighted by the user. Total similarity is calculated by the following formula:

$$Sim_{total}(e_1, e_2) = \sum_{i \in TS} p_i \times sim_i(e_1, e_2)$$

where: TS is the set of similarity types; p_i is the weight of the similarity types i.

Table 1 presents an example of the assignments of thresholds and weights by the user to have a syntactic, structural and semantic comparison.

- x, y and z are the weights values, such that x + y + z = 1
- Let x = 0.5, y = 0.25 and z = 0.25, the syntactic similarity measure has a high weight compared to structural and semantic similarity
- a, b and c are the thresholds values of name, visibility and abstraction, such as a + b + c = 1
- e, f and g are the thethresholds values of the name, type and visibility, such that e + f + g = 1
- k, l and m are the thethresholds values of the association, the association end, and the inheritance
- such that k + l + m = 1.

Table 1. Comparison criteria

Type of node	Weight	Criterion	Threshold
Class	x	Similarity of the name attribute	a
		Equal value of the visibility attribute	b
		Equal value of the isAbstract attribute	c
Attribute & Operation	y	Similarity of the name attribute	e
		Equal value of the type attribute	f
		Equal value of the visibility attribute	g
Association	z	Class element neighbor similarity	k
		Association end element neighbor similarity	l
		Inheritance element neighbor similarity	m

4. COMPARISON FUNCTIONS FOR SYNTACTIC, STRUCTURAL AND SEMANTIC SIMILARITY

The similarity assessment tool has a set of comparison rules, which have different aspects so that their matches and differences are better assessed [39]. Indeed, the comparison rules are expressed as follows:

- Syntactic similarities functions are used to measure the lexical similarity (names of classes, names of attributes, etc.) between compared elements
- Structural similarity functions are used to measure the similarity of properties (characteristics of attributes and operations, etc.) of the compared elements
- Semantic similarity functions are used to measure the similarity of the relations of the compared elements with their neighbors.

In the three types of comparisons, the concepts (class names, attribute names, operation names, and names of relationships between classes) are compared according to their syntactic similarity between two strings using their editing distance, and the domain of ontology, as well as other resources such as dictionaries (synonyms and hyponyms) [40]. This comparison is appropriate for measuring the similarity between the strings which may contain typos, acronyms, misspellings, etc. [41].

There are a number of measures proposed in the literature to measure the semantic similarity between two concepts. Some of these measures are based on the notion of information content (Resnik, 1995), while others are based on the length of the path [42]. These measures are simple and their success consists simply in measuring the conceptual distance between two concepts in the hierarchy of concepts [43].

4.1. Syntactic comparison functions

The syntactic similarity measure identifies the syntactic identity of two elements (classes, attributes, operations and relationships). Consequently, our evaluation of syntactic similarity is based both on a comparison of named elements similar to those defined in [44], by invoking a comparator of character strings for each pair of names to be compared. It searches for common substrings between two strings of two elements. It calculates the editing distance for each pair and returns a maximum similarity value [45]. Special characters and separators are ignored. Each comparator memorizes the elements it compares. The calculated similarity measures are identified in correspondence matrices to avoid recalculating them when comparing other auxiliary elements [46]. If the syntactic similarity measure of these elements has already been compared and they participate in other similarity measure then the existing comparator of these elements is consulted. The syntactic similarity measure is quantified using a set of similarity metrics defined as follows [47]:

a. Similarity measure between the names of two classes C_1 and C_2 and according to their visibility and their abstraction:

$$CSim_{syntax}(C_1, C_2) = s_{nc} \times compare_{nc}(C_1, C_2) + s_v \times compare_v(C_1, C_2) + s_a \times compare_a(C_1, C_2) \quad (1)$$

- s_{nc}, s_v represent arbitrary thresholds assigned to the similarity of classes names, visibilities and abstractions, respectively.
- $comparer_{nc}(C_1,C_2)$, $compare_v(C_1,C_2)$ and $compare_a(C_1,C_2)$ represent the comparison functions assigned to the similarity measure of the classes names, visibilities and abstractions, respectively.

b. Similarity measure between the names of two attributes A1 and A2, according to their visibility and their abstraction:

$$ASim_{syntax}(A_1, A_2) = s_{na} \times compare_{na}(A_1, A_2) + s_v \times compare_v(A_1, A_2) + s_t \times compare_t(A_1, A_2) \quad (2)$$

- s_{na}, s_v and s_t represent arbitrary thresholds assigned to the similarity of the names of the attributes, visibilities and types, respectively.
- $compare_{rna}(A_1,A_2)$, $compare_v(A_1,A_2)$ and $compare_t(A_1,A_2)$ represent the comparison functions assigned to the similarity measure of the attributes names, visibilities and type, respectively.

c. Similarity measure the between the names of two operations O1 and O2, according to their visibility and their type:

$$OSim_{syntax}(O_1, O_2) = s_{no} \times compare_{no}(O_1, O_2) + s_v \times compare_v(O_1, O_2) + s_t \times compare_t(O_1, O_2) \quad (3)$$

- s_{no}, s_v and s_t represent arbitrary thresholds assigned to the similarity of the names of the operations, visibilities and types, respectively.
- $compare_{rno}(O_1,O_2)$, $compare_v(O_1,O_2)$ and $compare_t(O_1,O_2)$ represent the comparison functions assigned to the measure similarity of operations names, visibilities and abstractions, respectively.

4.2. Structural comparison functions

Structural similarity measure that we propose focuses on syntactic similarity measure of all named elements between class diagrams. Indeed, structural similarity calculus uses the comparator of the classes names, attributes and operations of these classes. The result of the calculation will be qualified using a set of similarity metrics defined as follows:

- Similarity measure between the names of two classes C_1 and C_2, according to their visibility and their abstraction as determined by (1).
- Similarity attribute measure between two classes C_1 and C_2 is similarity measure between two sets of attributes, A_1 and A_2, respectively, defined as follows:

$$Sim_{att}(C_1, C_2) = \frac{\max[\sum_{k=1}^{|A1|} ASim_{syntax}(a_k, b_l)]}{|A2|} \quad (4)$$

$a_k \in A_1$ *and* $b_l \in A_2$, $|A_1| \leq |A_2|$. Similarity syntactic $ASim_{syntax}(a_k, b_l)$ between two attributes a_k and b_l is calculated on the basis of their syntactic similarity as defined in (2).

- Similarity operation measure between two classes C1 and C2 is similarity measure between two sets of operations, O1 and O2, respectively, defined as follows:

$$Sim_{op}(C_1, C_2) = \frac{\max[\sum_{k=1}^{|O1|} OSim_{syntax}(o_k, p_l)]}{|O2|} \quad (5)$$

$o_k \in O_1$ and $p_l \in O_2$, $|O_1|| \leq |O_2|$. Similarity syntactic $OSim_{syntax}(o_k, p_l)$ between two operations o_k and p_l is calculated on the basis of their syntactic similarity as defined in (3).

In abstract form, the calculation of structural similarity measure (C_1,C_2) is carried: p_c, p_a and p_o represent arbitrary weights assigned to the similarity measure of the classes names, attributes and operations, respectively.

$$Sim_{struct}(C_1, C_2) = p_c \times CSim_{syntax}(C_1, C_2) + p_a \times CSim_{att}(C_1, C_2) + p_o \times CSim_{op}(C_1, C_2) \quad (6)$$

The calculus of the structural similarity measure of two classes C_1 and C_2, is the sum of the syntactic similarity of the classes names, syntactic similarity of their attributes and syntactic similarity of their operations, respectively $CSim_{syntax}(C_1,C_2)$, $Sim_{att}(C_1,C_2)$ and $Sim_{op}(C_1,C_2)$. For example, the linked classes names are compared and matched to each other and the class attributes in a class diagram are compared and matched with the class attributes in the other diagram, etc.

4.3. Semantic comparison functions

Semantic similarity measure is determined by analyzing the direction in the elements and structure of diagrams. The relation of two classes implies the properties propagation from class mother to the child classes. A change in the direction of a relation between two classes, or replacement of relation type by another type, strongly modifies the semantics of the diagram [48]. We propose the calculation of the semantic similarity measure using three measures:

- The neighbor similarity measure which takes into account the comparison of the neighboring classes invokes a comparator of its structural similarity which was taken into account in the calculation phase for the matching of the structural similarity measure as defined in (2)
- The relationships similarity measure which takes into account the relationship name, the relationship type, the multiplicity, and the meaning of directed relationships
- The measure of similarity of inheritances which takes into account more particularly their numbers of roots, leaves, classes inheriting in a multiple way.

The semantic similarity measure is quantified using a set of similarity metrics defined as follows:

- The neighborhood similarity measure calculates the neighborhood similarity of two classes C_1 and C_2, having the two sets of neighbors V_1 and V_2, respectively, as follows:

$$Sim_{voisin}(C_1, C_2) = \frac{max[\sum_{k=1}^{|V1|} Sim_{struct}(m_k, n_l)]}{|V2|} \quad (7)$$

$m_k \in V_1$ *et* $n_l \in V_2$, $|V_1| \leq |V_2|$. The Simstruct similarity $Sim_{struct}(m_k, n_l)$ between two neighborhoods m_k and n_l is calculated on the basis of their structural similarity.

- Relation similarity measure between the compared classes and their neighbors Simrelation (C1,C2) is measured as weighted similarity of the comparison function of the association end type, the comparison function of the association name and the comparison function of the multiplicity. The relationship similarity measure is defined as follows:

$$Sim_{relation}\ (C_1, C_2) = s_{rt} \times compare_{rt}(m_k, n_l) + s_{rn} \times compare_{sr}(m_k, n_l) + s_{rm} \times compare_{sm}(m_k, n_l) \quad (8)$$

where s_{rt}, s_{rn} and s_{rm} represent arbitrary thresholds assigned to the types similarity of the association end, the names of the associations and multiplicities of the association end, respectively. $compare_{rt}(m_k, n_l)$, $compare_{rn}(m_k, n_l)$ and $compare_{rm}(m_k, n_l)$ represent the comparison functions assigned to the names of associations, the types and multiplicities of association end similarity measure, respectively. Semantic similarity measure measures the similarity of two classes C_1 and C_2, as similarity weighted by user-defined weights, is the sum of the neighborhood similarity measure, relationship similarity measure, and similarity measure inheritance [49]. The semantic similarity measure is defined as follows:

$$Sim_{semantic}(C_1, C_2) = \mathrm{p_v} \times Sim_{voisin}(C_1, C_2) + \mathrm{p_r} \times Sim_{relation}(C_1, C_2) + \mathrm{p_h} \times Sim_{héritage}(C_1, C_2) \quad (9)$$

where p_v, p_r and p_h represent arbitrary weights assigned to the neighborhood similarity measure, relationship similarity measure and inheritance similarity measure, respectively.

- The inheritance similarity measure is defined as follows:

$$Sim_{\ héritage}(C_1, C_2) = \frac{\max[\sum_{k=1}^{|H1|} Sim_{struct}(g_k, h_l)]}{|H2|} \quad (10)$$

$g_k \in H_1$ *and* $h_l \in H_2$, $|H_1| \leq |H_2|$. The similarity $Sim_{inheritance}(g_k, h_l)$ between two neighborhoods g_k and h_l is calculated on the basis of their structural similarity.

4.4. Weight setting

Our goal is to select the most appropriate weights automatically to detect matches, so that each class in a given class diagram corresponds to the most similar class in the other class diagram, based on the value of similarity. Indeed, we have carried out a series of experiments for collaboration in matters of weight. Each compared element must be assigned a weight which allows it to capture the similarity between these elements. The weight assignments of the constituents of the similarity measure are crucial for the accuracy of the metric. In this context, we consider that all close pairs with certain weights are similar, and that all less similar pairs are not a like. The weights of the similarity measures composed of n constituents are assigned values from 0 to 1 updated by 0.05, such as, $w_{x1} + w_{x2} + \ldots + w_{xn} = 1$. The weights are then assigned in the same way illustrated by the above pseudo-code [50]:

Algorithma 1: Evaluation of the semantic similarity measure

```
for pv = 0:0 ≤1 do
   for pr = 0:0 ≤ 1 − pv do
      for pg = 0:0 ≤ 1 − (pv + pr) do
        find Sim_semantic between UML graph classes G1 and G2
evaluate the matching between the UML graph classes of G1 and G2
      end for
    end for
end for
```

A pair of class diagrams was chosen at random. The weights are then assigned in the same way as the pseudo-code above. For each weight assignment, the similarity measure for each pair of classes in the two diagrams is calculated and added to the correspondence matrix. Each class in the unmatched class diagram in the other diagram is found. The weight setting that gives the best match result is used to match the other pairs in the diagram.

5. RESULTS AND DISCUSSIONS

In the previous section, we adapted the similarity measure between UML graphs to our own educational context. We have shown how our matching method is applied for the comparison of UML graphs and how its results are used to provide automatic corrections. We will now detail the actual implementation, put in order the different functionalities of our method and assess the quality of the results produced.

This matching method was developed for the needs of standardizing the formalism of the diagrams to be compared, syntactic validation, experimentation and reuse. We have chosen a representation of class diagrams in the UML meta model. This method is thus capable of measuring the similarities between several UML class diagrams, detecting differences, correcting errors and matching class diagrams. The results obtained (lists of syntactic and structural errors, identified differences, errors) in the form of a textual report, enabled us to carry out a summative evaluation starting from the sole achievement of the learner.

5.1. Assessment of class diagrams

The class diagrams modeled by learners were imported into our learning base, over three different exercises. Each exercise took place during a 1.5-hour continuous monitoring session on students second year Engineering computer science. The objective of these control sessions is to model class diagrams theoretically from a textual statement describing the speciations to be represented. The UML class diagrams constructed by the learners were corrected by the teacher for further analysis. We thus have at our disposal sixty-four class diagrams. Some diagrams, products may appear incomplete because some students have just had time to start the exercise or learners have not had enough time to do all the exercises requested during the control session.

We evaluated the relevance and the quality of the results produced by our method on a corpus of hundred class diagrams produced by the learners. From this corpus of diagrams, and a reference diagram for each exercise, we have chosen three exercises to configure and evaluate the system offline. We first improved the criteria involved in the calculation of the similarity functions and the general functioning from UML class diagrams constructed by the learners for the first exercise. Then, we tested and optimized the method on the second group of class diagrams from the second exercise. Some inconsistencies were identified and corrected, taking care not to degrade the quality of matching of the first test. Finally, the third group served to validate the method without any modification of the criteria.

5.2. Offline assessment

In this subsection, we present the results of the offline assessment in the form of histograms for the three exercises. The diagrams are numbered at the level of the abscissa axes. To study the intensity of the link that may exist between results of matching similarity obtained by a method score and the scores assigned by the teacher; we will study the linear correlation between these two variables. The linear correlation is then measured by calculating the linear correlation coefficient. This coefficient is equal to the ratio of their covariance and the not null product of their standard deviations.

To be able to measure the quality and relevance of the matching produced by the system, we compared the results found with those it actually finds. For the first exercise, we have found a correlation coefficient equal to 0.83. We note in Figure 4 some production does not conform to the results provided by the teacher. For the second exercise in Figure 5, we found a correlation coefficient equal to 0.98 and 0.96 for the third exercise as shown in Figure 6.

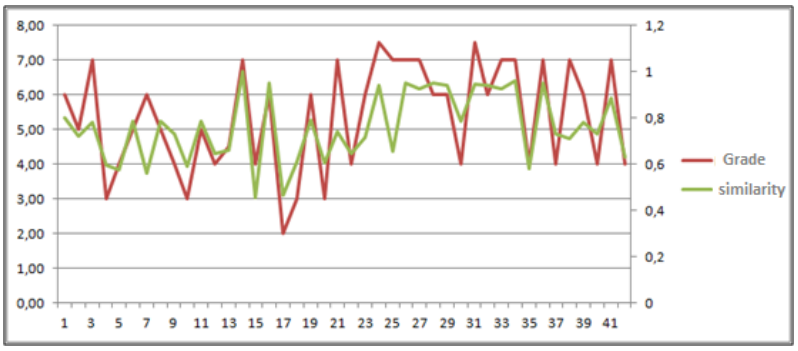


Figure 4. First exercise graph matching

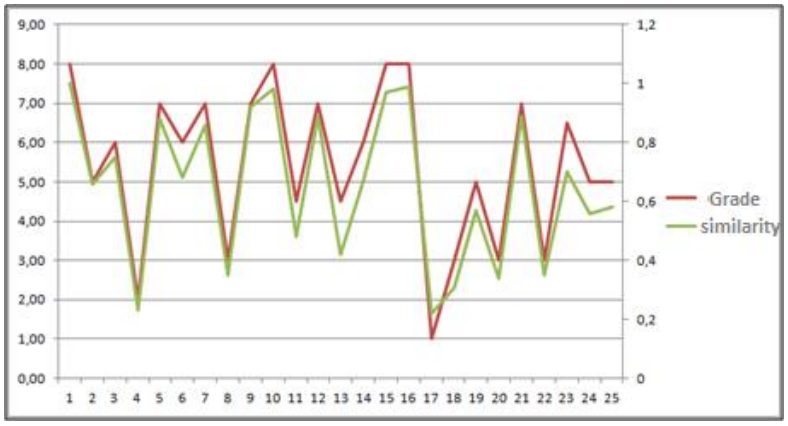


Figure 5. Second exercise graph matching

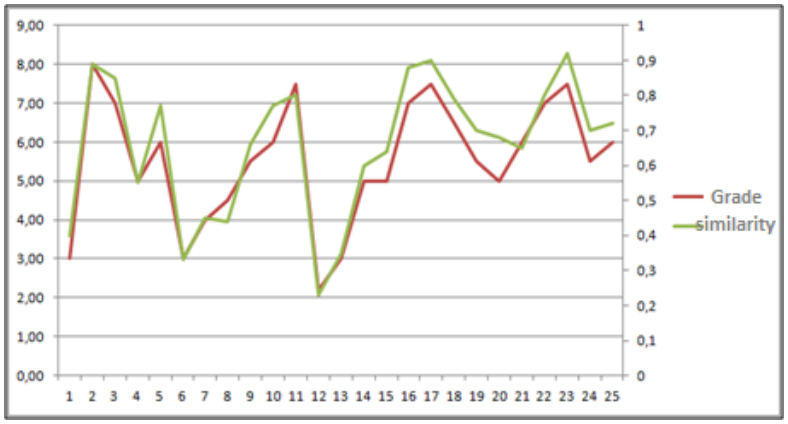


Figure 6. Third exercise graph matching

5.3. Experimental results and analysis

The application of the three quality measures to the results of all three exercises is shown in Table 2. The indicative calculation times of the system were performed on an Intel Xeon server with the Linux operating system and a processor clocked at 2.4 GHz. The results of the system show that on all compared diagrams, more than 80% of the matching provided conform to those expected whatever the diagrams compared. For 90% of the diagrams processed, the efforts required to correct the matching are minimal (Overall value greater than 0.85). The quality of the diagnosis is relatively good on simple and average problems (diagrams of the first second exercises). It can however be corrected and improved for more complex problems (diagrams of the third exercise). In particular, more than 85% of the results on average are

in line with those expected for more than 75% of the diagrams compared for the last exercice. The efforts required to correct errors and omissions in our method are greater in the last exercise, for just under 40% of matching diagrams, the Overall is greater than 0.85. However, for 85% of the diagrams, the Overall value is greater than 0.7 (a result which is still very acceptable in the field of diagram matching).

Table 2. offline evaluation result

Quality measure	Exercice 1 42 diagrams	Exercice 2 25 diagrams	Exercice 3 30 diagrams
Precision = 1	41	25	16
0.85 ≤ Precision < 1	1	0	20
0.7 ≤ Precision < 0.85	0	0	6
Recall = 1	41	20	5
0.85 ≤ Recall <1	0	2	22
0.7 ≤ Recall < 0.85	1	0	16
Overall = 1	41	20	5
0.85 ≤ Overall < 1	0	20	5
0.7 ≤ Overall < 0.85	0	0	22
0.55 ≤ Overall < 0.7	1	0	8
Times (min-max)	0,1 - 0,4 s	0,6 - 0,8 s	0,3 - 0,5 s

6. CONCLUSION

The problem to which we have tried to provide a solution relates to the summative evaluation of UML diagrams by a semi-automatic method. Indeed, although several systems have already tried to overcome this problem, they could not detect the errors made by the students, especially since the comparison is made only with a single solution, or a diagram of class can certainly be represented by several models. The objective of this paper then, was to develop a semi-automatic system, capable of correcting class diagrams through a comparison which is carried out thanks to the measurement of syntactic, structural and semantic similarity in order to find differences and specially to detect errors made by students. We have focused on the different methods of transforming UML diagrams into graphs, while having recourse to the different existing formalisms, and which we have been able to adapt to the problem linked to this article. We started with a study of evaluation as being a fundamental process in the validation of student achievement. At the end of this study, it was clear to us that the formative evaluation remains the best suited for the problem to which we are trying to provide a solution.

The results of the system show that 70% of the matches provided are in line with those expected on all diagrams compared. For 80% of the diagrams processed, the effort required to correct the matching is minimal (overall value greater than 0.85). The quality of the diagnosis is relatively good on simple and medium problems (diagrams of the first and the second exercises). In particular, more than 85% of the results on average are consistent with those expected for more than 75% of the graphs compared for the last exercice. The efforts required to correct errors is greater in the last exercice: for slightly less than 40% of the matched diagrams, the overall is greater than 0.85. However, for 85% of the diagrams, the overall value is greater than 0.7.

For research perspectives, one aspect to consider is that of the classification algorithm with an automatically generated learning base. This allowed us to carry out a summative and normative evaluation of the learners' productions. Generally speaking, we will divide the learning base into two main categories, class diagrams which are correct and class diagrams which are incorrect. Each diagram, of each category, is labeled according to its status and its degree of simplification. It is this same label which will allow us to carry out a summative evaluation of the learners' productions. Indeed, a learner's class diagram at a measure of maximum similarity of a labeled reference diagram will most likely belong to this class of diagrams. Another perspective is to apply our method to other types of structured models where formalism is defined. In particular, the method could be reused "directly" on other static models such as models have characteristics very close to UML class diagrams and can be considered as a subset of UML class diagrams.

REFERENCES

[1] S. Brau-Antony and C. Jourdain, "Évaluer la formation initiale des enseignants," *Évaluer pour former, Outils, dispositifs et acteurs*, p. 191, 2008.

[2] N. S. Shapiro, J. H. Levine, "Creating Learning Communities: A Practical Guide to Winning Support, Organizing for Change, and Implementing Programs," *Jossey-Bass Higher and Adult Education Series, ERIC*, 1999.

[3] J. A. Gottfried, et al., "Appetitive and aversive olfactory learning in humans studied using event-related functional magnetic resonance imaging," *Journal of Neuroscience*, vol. 22, no. 24, pp. 10829-10837, 2002.

[4] L. Suskie, "Assessing student learning: A common sense guide," *John Wiley & Sons*, 2018.
[5] C. M. Chen and Y. L. Li, "Personalised context-aware ubiquitous learning system for supporting effective English vocabulary learning," *Interactive Learning Environments*, vol. 18, no. 4, pp. 341-364, 2010.
[6] A. L. Powers, "An evaluation of four place-based education programs," *The Journal of Environmental Education*, vol. 35, no. 4, pp. 17-32, 2004.
[7] R. D. Crick, et al., "Evaluating the wider outcomes of schools: Complex systems modelling for leadership decisioning," *Educational Management Administration & Leadership*, vol. 45, no. 4, pp. 719-743, 2017.
[8] Y. Xiao and J. Hu, "Assessment of optimal pedagogical factors for Canadian ESL learners' reading literacy through artificial intelligence algorithms," *International Journal of English Linguistics*, vol. 9, no. 4, pp. 1-14, 2019.
[9] M. Samarakou, et al., "An Open Learning Environment for the Diagnosis, Assistance and Evaluation of Students Based on Artificial Intelligence," *International Journal of Emerging Technologies in Learning*, vol. 9, no. 3, pp. 36-44, 2014.
[10] M. M. Patchan, et al., "Accountability in peer assessment: examining the effects of reviewing grades on peer ratings and peer feedback," *Studies in Higher Education*, vol. 43, no. 12, pp. 2263-2278, 2018.
[11] M. Tanana, "Evaluation formative du savoir-faire des apprenants à l'aide d'algorithmes de classification: application à l'électronique numérique," Thèse de doctorat, INSA de Rouen, 2009.
[12] J. C. Muñoz-Carpio, et al., "Framework to Enhance Teaching and Learning in System Analysis and Unified Modelling Language," in *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 2018, pp. 91-98.
[13] C. Larman, "Applying UML and patterns: an introduction to object oriented analysis and design and interative development," *Pearson Education India*, 2012.
[14] N. Medvidovic, et al., "Modeling software architectures in the Unified Modeling Language," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 1, pp. 2-57, 2002.
[15] D. Dranidis, "Evaluation of StudentUML: an Educational Tool for Consistent Modelling with UML," in *Proceedings of the Informatics Education Europe II Conference*, 2007, pp. 248-256.
[16] E. Ramollari and D. Dranidis, "StudentUML: An educational tool supporting object-oriented analysis and design," in *Proceedings of the 11th Panhellenic Conference on Informatics*, 2007, pp. 363-373.
[17] D. Dranidis, et al., "Learning and Practicing Systems Analysis and Design with StudentUML," in *Proceedings of the 7th Balkan Conference on Informatics Conference*, 2015, pp. 1-8.
[18] P. Suraweera and A. Mitrovic, "An intelligent tutoring system for entity relationship modelling," *International Journal of Artificial Intelligence in Education*, vol. 14, no. 3, pp. 375-417, 2004.
[19] S. Ohlsson, "Constraint-based student modeling," in *Student modelling: the key to individualized knowledge-based instruction*, pp. 167-189, 1994.
[20] P. Suraweera and A. Mitrovic, "KERMIT: A constraint-based tutor for database modeling," in *International Conference on Intelligent Tutoring Systems*, pp. 377-387, 2002.
[21] L. Auxepaules, et al., "A diagnosis method that matches class diagrams in a learning environment for object-oriented modeling," in *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, 2008, pp. 26-30.
[22] Dominique P. Y., et al., "Diagram, a learning environment for initiation to object-oriented modeling with uml class diagrams," *Journal of Interactive Learning Research*, vol. 24, no. 4, pp. 425-446, 2013.
[23] F. Drewes, et al., "Graph transformation modules and their composition," in *International Workshop on Applications of Graph Transformations with Industrial Relevance*, 1999, pp. 15-30.
[24] K. Hölscher, et al., "On translating UML models into graph transformation systems," *Journal of Visual Languages & Computing*, vol. 17, no. 1, pp. 78-105, 2006.
[25] A. Outair, et al., "Towards a semi automatic assessment of UML diagrams by graph transformation," *2014 International Conference on Multimedia Computing and Systems (ICMCS)*, 2014, pp. 668-673.
[26] A. Outair, et al., "Towards an automatic evaluation of UML class diagrams by graph transformation," *International Journal of Computer Applications*, vol. 95, no. 21, pp. 36-41, 2014.
[27] A. Outair, et al., "Towards an Automatic Evaluation of UML Class Diagrams by Measuring Graph Similarity," *Journal of Theoretical and Applied Information Technology*, vol. 95, no. 4, pp. 928-935, 2017.
[28] G. L. Miller, "Graph isomorphism, general remarks," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 128-142, 1979.
[29] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151-158.
[30] S. Jouili and S. Tabbone, "Graph matching based on node signatures," in *International Workshop on Graph-Based Representations in Pattern Recognition*, 2009, pp. 154-163.
[31] M. Zaslavskiy, et al., "A path following algorithm for the graph matching problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2227-2242, 2008.
[32] P. Manuel and S. Klavzar, "A general position problem in graph theory," *Bulletin of the Australian Mathematical Society*, vol. 98, no. 2, pp. 177-187, 2018.
[33] Z. Wang, et al., "Key technology research on user identity resolution across multi-social media," in *2015 International Conference on Cloud Computing and Big Data (CCBD)*, 2015, pp. 358-361.
[34] M. Al-Rhman Al-Khiaty and M. Ahmed, "UML class diagrams: Similarity aspects and matching," *Lecture Notes on Software Engineering*, vol. 4, no. 1, pp. 41-47, 2016.
[35] T. I. Lin, et al., "Robust mixture modeling using the skew t distribution," *Statistics and computing*, vol. 17, no. 2, pp. 81-92, 2007.

[36] D. Kuang, et al., "SymNMF: nonnegative low-rank approximation of a similarity matrix for graph clustering," *Journal of Global Optimization*, vol. 62, no. 3, pp. 545-574, 2015.
[37] Q. Zhang, et al., "Measure the structure similarity of nodes in complex networks based on relative entropy," *Physica A: Statistical Mechanics and its Applications*, vol. 491, pp. 749-763, 2018.
[38] J. Wehren, "Ein XMI-basiertes Differenzwerkzeug für UML-Diagramme," Dissertation Diplomarbeit, FG PI, FB12, University of Siegen, 2004.
[39] Y. Tsong, et al., "Development of statistical methods for analytical similarity assessment," *Journal of biopharmaceutical statistics*, vol. 27, no. 2, pp. 197-205, 2017.
[40] S. B. Chaouni, et al., "MDA based-approach for UML Models Complete Comparison," *arXiv preprint arXiv: 1105.6128*, 2011.
[41] M. Keshavarz and Y. H. Lee, "Ontology matching by using ConceptNet," *Proceedings of the Asia Pacific Industrial Engineering & Management Systems Conference*, 2012, pp. 1917-1925.
[42] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," *Proceedings of the 32nd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics*, 1994, pp. 133-138.
[43] R. Fauzan, et al., "Class Diagram Similarity Measurement: A Different Approach," in *2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE)*, 2018, pp. 215-219.
[44] W. W. Cohen, et al., "A comparison of string metrics for matching names and records," *Kdd workshop on data cleaning and object consolidation*, pp. 1-6, 2003.
[45] X. Gou, et al., "Multiple criteria decision making based on distance and similarity measures under double hierarchy hesitant fuzzy linguistic environment," *Computers & Industrial Engineering*, vol. 126, pp. 516-530, 2018.
[46] X. Yuan, et al., "Multi-similarity measurement driven ensemble just-in-time learning for soft sensing of industrial processes," *Journal of Chemometrics*, vol. 32, no. 9, 2018.
[47] N. A. Lester, et al., "You can take a noun out of syntax...: Syntactic similarity effects in lexical priming," in *Cognitive Science*, pp. 2537-2542, 2017.
[48] S. Patwardhan, "Incorporating dictionary and corpus information into a context vector measure of semantic relatedness," Dissertation, University of Minnesota, Duluth, 2003.
[49] M. Al-Rhman Al-Khiaty and M. Ahmed, "Similarity assessment of UML class diagrams using a greedy algorithm," *2014 International Computer Science and Engineering Conference (ICSEC)*, 2014, pp. 228-233.
[50] M. Al-Rhman Al-Khiaty and M. Ahmed, "Matching UML class diagrams using a Hybridized Greedy-Genetic algorithm," *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, 2017, pp. 161-166.

BIOGRAPHIES OF AUTHORS

Anas Outair received the engineer degree in computer science and engineering at Abdelmalek Essaadi University, Morocco. He received his master of telecommunication and system. He obtained his bachelor of computer science (BS) degree with honors in mathematics and computer from Sciences and technologies Faculty, Tangier Morocco, in June 2002. His research interests include learning assessment, UML class diagram metrics, software reuse, and soft computing.

Mariam Tanana received the engineer degree in computer science from the ENSI engineer school of Caen (France), in 1990. She received the Ph.D. in computer science from the INSA engineer school of Rouen (France), in 2009. Currently, she is a professor at the ENSA engineer school of Tangier (Morocco), UAE University. His research interests include the learner's assessment in eLearning.

Lyhyaoui Abdelouahid received the bachelor's degree in electrical engineering from Université Abdelmalek Essaâdi, Tetouan, Morocco, in 1992, the master degree signal system and radio-communication from Escuela Técnica Superior de Telecomunicaciones, Universidad Politécnica of Madrid, in 1996, and the Ph.D. degree from Universidad Carlos III de Madrid, Spain, in 1999. Between 2000 and 2003 he was a Visiting Professor in Signal Theory and Communications at Universidad Carlos III de Madrid, Spain. Currently, he is a professor at Ecole Nationale des Sciences Appliquées of Tangier, Abdelmalek Essaâdi University. His main research interests include statistical learning theory, neural networks and their applications in multimedia signal processing and education.