Full Paper

Introduction

Teaching involves imparting knowledge and skill about concepts to students through definitions, illustrations and descriptions. Concept based teaching involves organizing various units of study so that a student is able to observe and identify patterns of facts and concepts (Loh, 2015). However, it is often the case that there is a difference (gap) between what the student has understood about a concept and what was originally presented by the teacher. For example, in the field of Computer Science, when a teacher teaches the concept of algorithm, the teacher often does not know what the student has understood by "algorithm" is what exactly the teacher meant. Assessment is one way of estimating this concept gap, where a student's knowledge and skills are judged as objectively as possible. Quantifying concept gap continues to remain an important challenge in Engineering education.

Techniques for quantifying concept gaps have not received much attention in the field of education. The techniques currently used are mostly informal in nature and are assessment based such as examination, laboratory exercises, and oral interactions in which the instructor probes the student with questions the answers to which will provide a qualitative estimate about the gap. Sometimes, lack of confidence in the student can also be viewed as a sign of the presence of concept gap.

However, considerable attention is paid to minimize the concept gaps. This includes motivating students, demonstrating practical applications, using game based tools, increasing interactions during lectures, making knowledge of concepts relevant to examinations, etc.

We distinguish two types of concepts: subject concepts and problem solving concepts. Subject concepts are derived from textbooks. Problem solving concepts relate to solving problems using subject concepts. For example, in Electronic Circuit Theory, Kirchhoff's laws and Ohm's laws are classified as subject concepts while choosing the minimum number of loops in an electronic circuit and solving them using mathematical techniques relate to problem solving concepts. In this paper, we argue that there is a need for a methodology for estimating concept gap objectively and propose a technique based on a formal model of concepts using which the concept gaps are measured.

Goal

Concepts can be primitive or abstract, and explicit or implicit. The objective in teaching is to inculcate correct and explicit knowledge about the concepts in the student's mind. However, depending on the background knowledge, the knowledge about a concept acquired by each student will vary, and this can be seen when the students choose to use them in situations such as during a written examination, viva voce, seminar presentation, etc. Both in teaching and in assessment, the challenge is how to estimate the concept gap between the student's (often partial) understanding of a concept and the concept that was originally taught to the student.

Our goal in this research is to provide an ontology based method whereby one can compute a measure of concept gap (henceforth denoted as G) in an objective way. In order to make the measure realistic, we take into account the implication of context as well by introducing weights for each concept. The measure G will be useful in judging how far a student's solution to a given problem is correct thus enabling an assessor to perform partial assessment of an otherwise incorrect solution objectively. In will also be useful in distributing marks across questions and their sub parts more systematically. As a by-product of this approach, we also

will present a heuristic for extracting concepts that are relevant in teaching from a textbook, and discuss strategies for prioritizing them.

Background

The role of concepts in human understanding has been well studied in philosophy (Corcho, 2000), medicine (Colombo et al., 2010), engineering (Swartout et al., 1996), and education (Brusilovsky, 2004). The concept of ontology was introduced to support the use of formally represented knowledge(Kitani et al, 2008). An ontology is defined as an explicit specification of a conceptualization (Gruber, 1993). An ontology describes the concepts and relationships providing a specification of the meaning of terms used in the vocabulary (Gruber, 1995), (Chandrasekaran, et al., 2003), (Knowledge Systems, 2015). Considerable work done in the field of ontology has focussed on techniques for automatically identifying concepts and the relationships amongst themselves in real world applications (Hazman, 2011). Similarity between concepts has also been a well investigated notion and there exists several algorithms that compute similarity amongst concepts (Resnik,1999) (Thiagarajan et al., 2008). Use of conceptual modelling and ontologies has also yielded encouraging results in the field of education (Kitani et al., 2008).

We view teaching as a process that involves transmitting often abstract concepts to the students. These concepts are typically defined in textbooks using definitions and illustrations. The concept teaching is further strengthened by problem solving in real world applications. Abstract concepts involve several lower level concepts that have been already taught. While teaching an abstract concept C to a student, the teacher assumes that the student has already understood all the lower level concepts that were previously taught to the student and that the student will now correctly relate the lower level concepts in understanding the newly taught higher level concept C. It is quite likely that the concept as understood by the student is not exactly C, but rather somewhat a variant C'. For example, in Programming, the concept of *type* may be understood by a student differently from what is defined in textbooks and taught. Thus, it becomes necessary to know how far C' differs from C. We can compute this by measuring the similarity between C and C' and use it to compute a measure of the deviation of C' from C. In this paper, we propose a simple technique for computing a measure of deviation between concepts and illustrate it in the field of teaching Algorithms and Data Structure. The method is intuitive and easy to adopt for hand computing.

Approach

In our approach, we use an ontology O of all concepts that we extract from the textbooks for the curriculum. A subset P of the concepts from O are assumed to be taught to the student in a course. We then extract a set of concepts Q from a student's solutions (from his answer book). Next we compute the similarity between the two concept sets P and Q. The complement of the similarity will be taken as the measure of the concept gap between P and Q.

As said before, we distinguish between two types of concepts from a text based representation: explicit and implicit. An explicit concept is a concept that is defined explicitly, for example, using a definition. Thus, "Algorithm is a sequence of steps and Bubble sort is an algorithm" is taken to define two concepts, namely, Algorithm and Bubble sort, both explicitly. However, "Bubble sort is a sequence of steps" is taken to define Bubble sort as an algorithm implicitly without defining what an algorithm is. We come across this situation several times while teaching and assessing students' knowledge about concepts. One of the challenges students seem to have is the lack of understanding of explicit knowledge versus implicit knowledge. Thus, while quantifying similarity in concept understanding in students, explicit concepts will be given higher score and implicit ones lower score. When we compute the

score, we often have to deduce the explicit concept from implicit concepts where the number of deduction rules will inversely determine the score. Thus, for explicit definition the score is 1 and for implicit definition the score is 1/k where k is the number of inference steps required to deduce the implicit description to the explicit definition. The score is 0 if the concept cannot be inferred even implicitly.

Concept gap computation when applied to an abstract concept such as when described as an algorithm does not necessarily verify the underlying computations in the algorithm. Concept gap computation assumes a concept map (in the form of a tree) and the gap is computed with respect to the map. If the details of computations are needed to be taken into account in the gap computation, the concept map must include all the necessary lower level concepts. It should be noted that that verifying the correctness of an algorithm is known to be a hard task. If there are n nodes and m edges in the concept map, concept gap computation process will take time at least proportional to n+m.

Methodology for concept gap computation

Input A concept tree for a concept C and a description of the same concept as provided by a student. Let C' be the concept that is defined in the description provided by the student. We assume that each node in the tree has a weight wi attached to it.

Output A score that is a measure of the concept gap between C and C'.

Method

Rather than extracting C' from the description provided by the student, we use the description itself while computing the concept gap.

1. Consider a leaf node v0 of the tree. (Leaf node is considered to be at level 0). Let Cv0 be the concept associated with this node.
2. Examine the student's description and check if Cv0 is defined. If Cv0 is defined explicitly, then assign a full score of 1 to the node v0. Otherwise, Cv0 may be implicitly present. If k is the number of inference steps required to deduce the concept Cvo from the implicit description, then assign a score of 1/ (k+1). If neither explicit description nor the implicit description of Cv0 is present, then assign a score of 0.
3. Repeat this to every leaf node of the tree.
4. Compute the score for the next higher level (level 1) nodes as follows: Consider a node v1 at level 1. Consider all its children. Check if v1 is defined from the definitions of all its children explicitly in the student's description. If it is, then give a score 1. Otherwise, if it is implicitly present assign a score of 1/(k+1) where k is the number of deduction steps; else assign 0.
5. Repeat the above step for all level 1 nodes.
6. Repeat the above for all higher levels until the root is reached.
7. Compute the overall score as (s1*w1 + s2*w2 + … + sn*wm)/(w1+…+wn) where si is the score of a node vi, wi is the weight attached to vi and n is the total number of nodes in the tree.

Applications to Computing courses on Data Structures

Data structures and algorithms are interesting objects that have multiple layers of concepts that can be easily extracted and concept graphs built. We consider an example.

Abstract Data Type (ADT) An ADT is a non-basic type with a name and a set of operations performed on a collection of data held in a data structure within the type. The details of the data structure and the operations are totally hidden from the user of the type (Weiss, 1997).

There are several ways one can represent the above definition of ADT. For the purpose of measuring concep[illegible]n below.

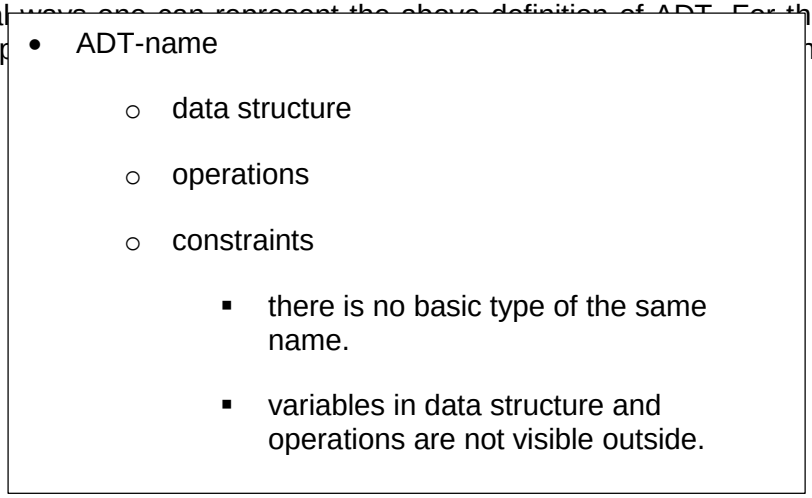

Figure 1. Concept tree for ADT definition.
(Note: The tree is shown as a nested structure textually.)

Similarly we can define a concept tree for an ADT called Stack as shown below.

ADT Stack This is defined as an ADT whose name is Stack with operations top ( ), push(x) and pop ( ) such that the variables used in them are not visible outside (Weiss, 1997).

We can now represent this ADT as follows.

- Stack
  - data Structure: An open ended stack of cells
  - operations
    - push (x) /* stores x in a cell and places it in the stack as the top most cell of the stack.*/
    - pop ( ) /* removes the top most cell of the stack and returns its content */
    - top( ) /* gives the content of the top most cell in the stack such that the content of the stack remains same after the execution of the operation */
  - constraint
    - Stack is not a basic type already defined.
    - Stack cells are not accessible from outside the type.

Figure 2. Concept tree for Stack ADT definition.

We can use the concept trees defined above to measure the concept gap for the corresponding descriptions provided by students.

Student's definition of ADT

Consider the description of ADT obtained from a student who sat an examination on Data Structures and Algorithms.

An abstract data type is defined as a data type which is not pre-built in a language. It has properties such that the user may create and modify the data type according to his own needs. The structure of these data types is not regulated by the language, but the users.

In order to compute the concept gap between this definition and the definition in Figure 1, we interpret the student's definition using the tree in Figure 1. (The student's description is shown in italics below.)

1. An abstract data type is defined as a data type which is not pre-built in a language.
    a. Score = 1.0 since it matches with one concept node in Figure 1.
2. It has properties such that the **user may create and modify** the **data type** according to his own needs.
    a. For this, there is no explicit match with any concept node in Figure 1. However, implicit match is possible. From **user may create and modify,** we can infer **operations** which is a node in the tree in Figure 1. This is written as: **user may create and modify → operations**.
    Thus, we assign a score of 0.5. to the node **operations**.
    b. Similarly**, data type** → data structure. So, score = 0.5
3. The **structure of these data types is not regulated by the language, but the users.**
    a. **data types is not regulated by the language, but the users → user defined operations**. Score = 0 since it is a repetition.

The results of interpreting the student's description using the concept tree in Figure 1 can be shown in the concept tree as follows.

- ADT-name [0.0 *w1]
    - data structure [0.0 * w2]
    - operations [0.5 * w3]
    - constraints [0.0 *w4]
        - there is no basic type of the same name. [1.0 *w5]
        - variables in data structure and operations are not visible outside.[0.0 * w6]

Figure 3. Computing similarity in each concept

We now can compute the overall similarity as [(0*w1 + 0*w2 + 0.5*w3 + 0.0*w4 + 1.0*w5 +0.0*w6)/(w1+…+w6)]. Letting, wi = 1 for all i, we have overall similarity = 0.25. Thus, 1- 0.25 = 0.75 is the computed measure of the difference between the concept that was taught to the student and the concept that the student understood.

We will now show how we can get a measure of the concept gap for programs. Consider the program for the algorithm for bubble sort.

```
void bubble_sort(int iarr[], int num) {
  int i, j, k, temp;
  for (i = 1; i < num; i++) {
    for (j = 0; j < num - 1; j++) {
      if (iarr[j] > iarr[j + 1]) {
        temp = iarr[j]; iarr[j] = iarr[j + 1];  iarr[j + 1] = temp; }}}}
```

Figure 4. Bubble sort in C code.

The concept tree for bubble sort as written above is given below.

- Bubble sort
  - Parameters
    - Array and size
  - Return type
  - Outer Loop
    - Initialization and exit condition
    - Inner loop
      - Swap adjacent locations in the array
      - Condition for swap
  - Variable declaration

Figure 5. Concept tree for bubble sort.

We can now use this concept map to interpret the student program below in Figure 6 and compute the concept gap.

```
bubble_sort(a[],n){
int j = n-1;
boolean did_swap = true;
while ( ( j >= 0) && (did_swap)){
  did_swap = false;
  for ( k= 1 to j) {
      if (a[k] > a[k+1])
           then swap (a[k], a[k+1]);
           did_swap = true;
  }
j = j + 1; /* Error here; it should be j = j -1 */
}
```

Figure 6. Bubble sort pseudo code from a student answer.

Figure 7 shows the interpreted result.

- Bubble sort
  - Parameters
    - Array and size [1.0*w1]
  - Return type [NA] /* Not applicable */
  - Outer Loop
    - Initialization and exit condition [1.0*w2]
    - Inner loop
      - initialization and condition [1.0*w3]
      - Swap adjacent locations in the array [1.0*w4]
      - Condition for swap [1.0*w5]
  - Variable declaration [1.0*w6]

Figure 7. Computing similarity in each concept.

Similarity measure = [(1.0*w1 + 1.0*w2 +1.0*w3 +1.0*w4 +1.0*w5) / (w1+…+w5)]. Assuming wi=1, similarity = 1.0 and thus, concept gap = 0.

Note that there is an error in the pseudo code in Figure 6, and yet it does not affect the similarity value because the error only affects the computational step but not the conceptual component as defined by our concept graph.

Discussion

We applied the above methodology for estimating concept gaps for students' answers from Data Structures and Algorithms course from the field of Computer Science. Intuitively, G captures the notion of semantic gap between the reference concepts (from the textbook) and the concepts as used by the student. It is reasonable to assert that the partial assessment provided by a human assessor is also a measure of the semantic gap.

Our methodology can be useful in objectively formulating guidelines for evaluating student performance in various assessment schemes. For example, in written examinations, incorrect answers can be more objectively marked. In assignments, the work submitted by a student can be assessed at multiple levels: methodology level, procedural level, implementation level, and performance level.

Conclusion

Reactive assessment techniques, such as viva voce, can be more effective when the questions asked at any time t depends on the performance until the previous moment, namely, (t-1). Using our methodology, the assessor can estimate the conceptual deviation until the moment t, and can then use it to choose the next set of concepts to assess the student.

The technique we have provided are easily implementable and can be used flexibly to customize teaching strategies taking individual student's strengths and limitations. Our approach can be used not only for assessment at any time, but also for helping a student acquire the knowledge the student lacks about a concept thus aiding the student in the overall learning process.

Our work raises a more fundamental question philosophically. The methodology we have proposed can be generalized to study the deviational behaviour in several systems such as hardware systems, human task execution behaviour, and the performance behaviours of even social systems such as a groups and teams.

References

Resnik,P.(1999) "Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language", Journal of Artificial Intelligence Research, Volume 11, pp. 95-130.

Kitani, N., and Yukita,S. (2008) The Educational Uses of Mathematical Ontology and the Searching Tool, 38th ASEE/IEEE Frontiers in Education Conference.

Brusilovsky P., Sosnovsky S., Shcherbinina O. QuizGuide: Increasing the Educational Value of Individualized Self-Assessment Quizzes with Adaptive Navigation Support. In Janice Nall and Robby Robson (eds.) Proceedings of E-Learn 2004. Washington, DC, USA: AACE, 2004, 1806-1813.

Colombo, G., Merico, D., Boncoraglio, G., De Paoli, F., Ellul, J., Frisoni, G., et al. (2010). An ontological modeling approach to cerebrovascular disease studies: The NEUROWEB case. *Journal of Biomedical Informatics, 43*(4), 469-484.

T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220, 1993.

Hazman, M., El-Beltagy, Rafea, A., Survey of Ontology Learning Approaches, International Journal of Computer Applications, vol 22, No 9, May 2011.

Oscar Corcho and Asuncion Gomez-Perez, A Roadmap to Ontology Specification Languages, in Rose Dieng and Olivier Corby (eds.), *Knowledge Engineering and Knowledge Management. Methods, Models and Tools*, Springer, Berlin, 80-96, 2000.

Swartout,B., Patil, R., Knight, K., and Russ,T. (1996) Toward distributed use of large-scale ontologies, In Proceedings of the Symposium on Ontological Engineering of AAAI, pp 138 – 148, AAAI.

Thiagarajan, R., Manjunath,G., and Stumptner, M., Computing Semantic Similarity Using Ontologies ISWC 08, the International Semantic Web Conference (ISWC), 2008, Karlsruhe, Germany.

Loh, A., What is Concept-Based Education, Retrieved October 19, 2015 from http://www.brainy-child.com/article/concept-based-education.shtml

Gruber, T.R. (1995) Toward principles for the design of ontologies used for knowledge sharing , *Int. J. Human-Computer Studie*s, 1995,pp. 907-928.

Chandrasekaran, R., Josephson, J.R., & Benjamin, V.R. (2003) “What are ontologies, and why do we need them?”, *IEEE Intelligent Systems*, 14,2003, pp.20-26.

Knowledge Systems, AI Laboratory, Stanford University, "What is an ontology?”, Retrieved October 19, 2015 from http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

Weiss, M.A. (1997) Data Structures and Algorithm Analysis in C, Addison Wesley, Menlo Park, California.