Assessing Procedural Knowledge in Free-text Answers through a Hybrid Semantic Web Approach

Eric Snow, Chadia Moghrabi, and Philippe Fournier-Viger
Département d’informatique
Université de Moncton
Moncton, New Brunswick, Canada
{eric.snow, chadia.moghrabi, philippe.fournier-viger}@umoncton.ca

Abstract—Several techniques have been proposed to automatically grade students’ free-text answers in e-learning systems. However, these techniques provide no or limited support for the evaluation of acquired procedural knowledge. To address this issue, we propose a new approach, named ProcMark, specifically designed to assess answers containing procedural knowledge. It requires a teacher to provide the ideal answer as a semantic network (SN) that is used to automatically score learners’ answers in plain text. The novelty of our approach resides mainly in three areas: a) the variable granularity levels possible in the SN and the parameterizing of ontology concepts, thus allowing the students free expression of their ideas; b) the new similarity measures of the grading system that give refined numerical scores; c) the language-independence of the grading system as all linguistic information is given as data files or dictionaries and is distinct of the semantic knowledge of the SN. Experimental results in a Computer Algorithms course show that the approach gives marks that are very close to those of human graders, with a very strong (0.70, 0.79, and 0.79) positive correlation.

Keywords—procedural knowledge; computer-assisted assessment; free-text answers; ontology; semantic networks; similarity measures.

I. INTRODUCTION

Assessing the students’ learning in an e-learning environment often relies on multiple choice or fill-in-the-blank questions, which only trigger the lowest level (Knowledge) of Bloom’s taxonomy of knowledge acquisition [1]. Several attempts have been made to incorporate open-ended questions in online assessment, which would favour triggering the higher levels of Bloom’s taxonomy (Synthesis and Evaluation) in the students’ learning. A variety of techniques have been used to develop e-learning environments that can automatically grade free-text answers, such as Natural Language Processing (NLP) [2]–[4], statistical techniques [5]–[7], or Information Extraction (IE) [8], [9]. Although human intervention is still required in the grading process, automatic assessment is much faster and is done more objectively than manual scoring. Techniques for automatic assessment perform well on texts containing factual knowledge (that is, static description of events, places, etc.). However, they suffer from an important limitation: they generally provide no or limited support for the evaluation of procedural knowledge (e.g. [10], [11]). For several domains, it is not only crucial to evaluate the facts stated in a text, but also the temporal relationships between the steps of the procedure expressed through the statements. If these relationships are ignored, texts containing the expected keywords (even in the wrong order) are still awarded high marks. For example, in Computer Science, to exchange the contents of two variables \( a \) and \( b \), the three steps: \( c := a \), \( a := b \), and \( b := c \) have to be stated in the student answer in this order. Procedural knowledge is very common in many domains, such as cooking, call center troubleshooting and mechanical repairs.

To address this issue, we propose a novel approach, named ProcMark, specifically designed for the assessment of free-text answers containing procedural knowledge. The system requires a teacher to provide an ideal answer for each question as a semantic network (SN), based on a course ontology. The expected student answer is a free text, around a paragraph in length, explaining the operation of a computer algorithm they learned. The system utilizes the semantic network to grade answers automatically. Our system bears some similarity to OeLE’s [10] grading algorithm, with the addition of three novel similarity measures that allow the assessment of procedural relationships between steps described in the texts. The semantic network representation of ideal answers is akin to MultiNet Working Bench (MWB) [11], with our system having more parameterized concepts. However, these latter systems are not specifically designed to assess procedural knowledge.

The contributions of our system are multifaceted. First, to allow the representation of procedural knowledge, we introduced in [12] the notion of functional concepts that operate like computer programming functions or procedures. These functional concepts can be nested (decomposed into more specific concepts) and reused (multiple function calls). In this new version, we define an expanded ontology offering general algorithmic knowledge, such as flow control structures, conditions, etc., and use SNs to represent answers.

Second, to express and handle different granularity levels of knowledge in the ideal answer’s semantic network, we chose that all knowledge representation be parameterized as concepts, namely functional concepts, flow control struc-
tutes, conditions, relations, attributes, and literals. These design decisions also give students more liberty by allowing them to express their answers in more or less detail.

Third, to support these levels of granularity in students’ answers, we propose an improved grading algorithm. It includes three novel similarity measures: part-whole similarity, which considers the nesting and composition of ontology concepts; textual proximity, which takes into account the distance between a concept and its related concepts as expressed in the students’ answers; and temporal ordering, which considers the relative order between procedure steps.

Finally, to evaluate our approach, we applied it to questions in a course on computer algorithms. Experimental results show that the marks given by our system are very close to those given by human graders, with a very strong (0.70, 0.79, and 0.79) positive correlation.

The remaining of this paper is organized as follows. Section II surveys automatic free-text assessment systems. Section III presents our approach, followed by our results in Section IV. We conclude the paper in Section V with a brief discussion and our final remarks.

II. LITERATURE REVIEW

This section surveys previous and ongoing research in automatic short-answer assessment. Other systems have also been developed for longer essay scoring, but will not be examined here. A detailed review can be found in [13].

Some systems require teachers to provide training sets of marked student answers. For example, Auto-marking [2] uses pattern-matching to compare students’ answers to a training set of marked student answers. Comparative experiments were conducted using Inductive Logic Programming (ILP), decision tree learning and Naive Bayesian learning. This system obtained 88% of exact agreement with human grading in factual questions about a Biology course.

Bayesian Essay Test Scoring System (BETSY) [3] uses Naive Bayes classifiers to search for specific features in students’ answers. In a Biology course, it achieved up to 80% accuracy. In [4], a combination of NLP and Support Vector Machines (SVM) is used to classify answers into two classes (above/below 6 points out of 10). It obtained an average of 65% precision rate (the only reported metric).

Other systems require a set of reference texts. For example, Research Methods Tutor [5] uses Latent Semantic Analysis (LSA) to compare the students’ answers to a set of expected answers. If the student answers incorrectly, the system guides the student into obtaining the right answer. The Willow system [6] requires several unmarked reference answers for each question. It also uses LSA to evaluate students’ answers written in English or Spanish. In a Computer Science course, it achieved on average 0.54 correlation with the teacher’s grading. A system currently in use at the University of Witwatersrand [7] uses LSA and clustering techniques. It achieves between 0.80 and 0.95 correlation with the teacher’s grading. Their system will not process correctly “The left most node of the right subtree” and “The right most node of the left subtree”. Our system recognizes these differences as it assigns different attributes to the respective nodes and subtrees in the SN.

Many of the previous systems use machine learning techniques, which explains the large number of examples for the training set. Some systems compare students’ answers to an ideal answer supplied by the teacher. For instance, Automated Text Marker [8] uses pattern-matching. It has been tested in courses on Prolog programming, psychology and biology. Although Prolog programming is one of the domain applications, the knowledge assessed is factual rather than procedural. Automark [9] uses Information Extraction techniques to grade students’ answers by comparing them to a mark scheme template provided by the teacher. It achieved 94.7% agreement with human grading for six of the seven science-related factual questions asked on a test exam.

The OeLE [10] system uses a combination of NLP techniques and Semantic Web ontologies to assess the level of understanding of the students. The teacher-provided ideal answer is annotated against the course ontology and compared to the (similarly annotated) students’ answers during the assessment. The researchers expect a deeper understanding of the text than statistical techniques. The OeLE system has been used in two online courses.

The MultiNet Working Bench [11] system also attempts a deeper understanding of the text with the help of semantic networks. A separate graphical tool is used to represent the students’ answers visually. It compares the semantic network extracted from the student answer to a reference semantic network submitted by the teacher. Verified parts of the network are displayed in green, while wrong or unverified parts (not supported by logic inference) are displayed in red. MWB has been applied to the evaluation of Java, Prolog, Scheme and C programming by processing program code input by the student.

Pathfinder Network analysis is used in [14] to score concept maps hand-sketched by students to illustrate relationships between terms or concepts. Spatial position of terms is taken into account when automatically assessing students’ knowledge. Pathfinder measures the pair-wise distance between concepts in pixels (from a rendering of the sketch in the ALA-Mapper software). We define textual proximity as a function of the distance between words or expressions of the students’ concepts in their texts.

A hierarchical similarity measure is introduced in [15] to calculate the distance between nodes in a directed acyclic graph (DAG). The measure is used for text categorization, where the categories form a DAG. This measure is more refined than the ontological proximity measure used in our approach and in [10]. However, our approach relies as well on a recursive part-whole similarity measure to assess procedural knowledge more precisely.
Semantic similarity measures are also used in [16], a move away from bag-of-words (BOW) approaches. This approach tries to align dependency graphs, which represent relationships between concepts, in student and teacher answers, in order to calculate the grade. It was applied in a Data Structures course at the University of North Texas.

To our knowledge, no system is specifically designed to assess procedural knowledge in free-text answers, although such a system would be helpful in an e-learning environment for scientific and technical domains, where procedural knowledge is very common.

III. THE ProCMARK SYSTEM

In this section, we present our system, which is designed to assess free-text student answers containing procedural knowledge. The system works in four phases, which are explained in detail in the next subsections. In the first phase, a teacher builds a SN for each question’s ideal answer by using the course ontology. In the second phase, the system annotates a student’s answer by using NLP techniques. It matches the concepts occurring in the ideal answer to words or expressions in the student’s answer. The third phase links the concepts found in the annotated answer to nodes of the ideal answer’s SN, thus forming the specific SN of the student’s answer as an overlay. Finally, during the SN traversal, the system calculates the similarity measures that are used by the grading algorithm to give a numerical mark.

A. Representing Domain Knowledge

The first phase consists of defining the course ontology to represent the procedural knowledge and then instantiating part of it for each question as a SN of the ideal answer.

1) Course Ontology: The course ontology, in a Computer Algorithms course, contains the usual control structures, operators, conditions, etc. Our ontology contains 131 entries, organized as a hierarchy. It is encoded in the Web Ontology Language (OWL), which allows the use of graphical editors such as Protégé, and facilitates its reuse and sharing with other teachers. Fig.1 shows a partial concept class hierarchy. The course ontology includes 10 categories of concepts (cf. Table I) that are used for all exam questions. It is possible to define new concepts and categories, as needed, when more questions are added. A more detailed explanation of these concepts and their categories is given in the next section.

2) Semantic Network of Ideal Answer: The ideal answer for each question is given by the teacher in the form of a semantic network (SN). The SN is also encoded in OWL and it imports the course ontology. The SN’s nodes are instances of concepts defined in the course ontology. This network is a directed connected graph. Its root is always a functional concept representing a computer algorithm.

A functional concept (FC) represents a global procedure, a sequence of subprocedures, or individual steps accomplishing a given task. For example, one of the questions asked for an exam was to explain the operation of Dijkstra’s shortest path algorithm in plain text. For this question, the ideal answer contains 77 nodes and 80 arcs. The nodes are instances of concepts of the course ontology. Let us consider the algorithm’s pseudocode, using some ontology concepts:

\begin{verbatim}
procedure Dijkstra
    While NOT AllVerticesAreVisited
        ChooseVertexV
        MarkVAsVisited
        ForEach Vertex W Adjacent to V
            ModifyDvOfW
            ModifyParentOfW
        end
    end
end
\end{verbatim}

For every procedure or subprocedure, we create a corresponding FC: \textit{Dijkstra}, \textit{ChooseVertexV}, \textit{MarkVAsVisited}, \textit{ForEach Vertex W Adjacent to V}, \textit{ModifyDvOfW} and \textit{ModifyParentOfW}. The FCs allow for a variable granularity of the algorithm description. When implementation details are needed, subprocedures (FCs) are

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Concept categories} & \textbf{Example concepts} \\
\hline
FlowStructure & ForEach, While \\
Iterative & AndThen, OrThen \\
Sequential & If \\
Selective & \\
Field & Visited, Dv, Parent \\
Element & Vertex, Edge \\
Set & Vertices, Edges \\
Quantifier & ForAll, ThereExists \\
Operator & Addition, Subtraction \\
Arithmetic & AND, OR, NOT \\
Logical & EqualTo, GreaterThan \\
Comparison & Dereference \\
Member & \\
Condition & AllVerticesAreVisited, VIsVisited \\
Variable & V, W \\
FunctionalConcept & Dijkstra, B+Search, ChooseVertexV \\
Literal & True, False \\
\hline
\end{tabular}
\caption{Categories of concepts and examples}
\end{table}
in turn further decomposed. This decomposition only needs to be specified once and can then be reused, much like computer programming functions and procedures are defined once but can be called multiple times.

Functional concepts form only one category of concepts in our system’s ontology. As mentioned, we define several other categories of concepts, which represent usual programming constructs. In particular, we give special consideration to the flow structure category, as it allows the temporal ordering of concept instances with two sequential concepts, AndThen and OrThen. These concepts allow instances of FCs to be ordered temporally. For example, ChooseVertexV <AndThen> MarkVAsVisited indicates that these concepts are required in that order. Alternatively, ModifyDvOfW <OrThen> ModifyParentOfW indicates that these concepts can be done in any order.

Another key concept category is the operator. For example, it defines the concept of logical negation (NOT), which allows modifying the type of iterative flow structures. For example, While NOT AllVerticesAreVisited <do> X can be transformed into Until AllVerticesAreVisited <do> X. It also allows specifying all conditions in the positive form (VisVisited) and negating them easily (NOT VisVisited).

The variable is another important concept category. These concepts are used to emulate variables of computer programming. Their treatment is discussed in Section III-C.

In addition, we promoted relations, attributes and literals to full-fledged concepts (reified to first-class objects) in order to allow our algorithmic ontology to be as flexible and parametric as possible. This allows relations to be nested and to take not only concepts as their domain or image, but also other relations, attributes and literals. We name these newly promoted concepts: relation-concepts, attribute-concepts, and literal-concepts. The concepts highlighted in italics in Tables I and II are relation-concepts. Fig. 2 shows an example of the use of attribute-concept Visited and literal-concept True, which are connected by the value relation.

Using relation-concepts imposes the creation of new relations to connect all concepts uniformly. In the remainder of our paper, we simply refer to these new relations as relations, to distinguish them from relation-concepts. Our course ontology thus contains 15 relations (cf. Table II). The relations are used to label the edges of the SN, and hence participate only indirectly in the grading process. Fig. 3 shows an example of the nesting of relation-concepts (While, NOT, AndThen, OrThen) and of relations (condition, operand, do, doBefore, doAfter).

We now present four relations that need special consideration, as they are used for similarity measures and for the treatment of sequences of procedural knowledge. The decomposition relation is used in the SN to join a FC to the root of the subgraph detailing it. It is used to zoom in on more details if needed in the student answer. For example, Fig. 4 shows how the FC ChooseVertexV is decomposed for Dijkstra's algorithm into the concepts Assignment, V, and UnvisitedVertexOfMinDist, which represents assigning to the variable V the unvisited vertex with total minimum distance. Further decomposition of the UnvisitedVertexOfMinDist FC is not shown in this figure.

We have previously explained the behavior of AndThen and OrThen. The relations doBefore and doAfter are used...
in conjunction with the *AndThen* relation-concept to specify the required sequence. For example, Fig. 3 shows the use of *AndThen*, *doBefore*, and *doAfter* to specify that vertex *V* has to be chosen before it is marked as visited.

The *do* relation is used in conjunction with *OrThen* to indicate that two concepts can be performed in any order. For example, the grayed area in Fig. 3 illustrates the use of *OrThen* and *do* to indicate that the field *Dv* of the node *W* can be modified before or after changing its parent. The *do* relation is also used with the *While* relation-concept to indicate the block of steps to be repeated. For example, Fig. 3 shows the use of *While* and *do* to indicate that the subgraph rooted at *AndThen* has to be repeated.

To specify more complex ordering relationships, *AndThen* and *OrThen* can be nested. Fig. 5 shows such nesting, which allows the four orderings: *ABCD*, *ABDC*, *CDAB* and *DCAB*.

**B. Annotating Students’ Answers**

In the second phase, the system annotates a student’s answer by using NLP techniques, namely: semantic networks, morphology analysis, synonymy and word sense disambiguation, tokenization, and stemming. It matches the concepts occurring in the ideal answer to words or expressions in the student’s answer. This phase encompasses the *Preparation* and *Matching* steps.

The students enter their answer in a natural language (French, in our case). Our system could easily be adapted to support other languages by providing the appropriate dictionaries. The students are encouraged to produce full sentences; pseudocode and explicit code structure, such as indentation and braces, should be avoided in answers.

The system uses two constructed French dictionaries. The first (D1) is a thesaurus containing 244 synonyms. The second (D2) contains 194 linguistic expressions associated to course ontology concepts. The system recognizes 262 verbs, 370 nouns, 155 adjectives, 141 adverbs, and 48 others words.

1) *Preparation*: Since 60% of the students in our experiments are non-native French speakers, we manually spell-checked the answers and made grammatical revisions before processing the texts. The texts are then tokenized, stemmed, and synonyms are resolved to their canonical form using D1. The stemmed, canonical tokens are placed in a single array for each answer. We experimentally determined that our algorithm performs better when the answer’s tokens are processed together, rather than one sentence at a time.

2) *Matching*: This step detects and matches the concepts of the teacher’s ideal answer in the students’ answers. In an ideal situation, the words and expressions in the students’ answers are automatically annotated using previous annotations stored in the knowledge base, D2. In the case where the SN’s concepts are not matched, the teacher has the option of adding an annotation, thus augmenting the system’s knowledge base with the corresponding linguistic expression (LE). These LEs are processed similarly to the student’s answers (see *Preparation*, above). For example, when looking for concept *Parent*, previous LEs could be “its parent” and “the parent”. Each of the stemmed, canonical tokens of the previous LEs are compared to the stemmed, canonical tokens of the student’s answer. If a new LE is found in the student’s answer, for example “the father node”, then a new LE is added to the knowledge base. Section III-D shows how the lexical similarity measure takes into account the processing level needed to match the LEs.

If an ideal answer’s concept *C* is not found in the student’s answer, the system looks for the ontological siblings of *C* first in the student’s answer. If these concepts are not found, then the system looks for the parents, and finally the children of *C*. In case the matching fails, the concept *C* is marked as absent, otherwise it is marked as a fuzzy match. As will be seen in Section III-D, fuzzy matches and absent concepts negatively impact the student’s grade.

The matching algorithm adopts the strategy of “wait-and-see” regarding the problems of synonymy and ambiguity. A given concept can be found in multiple locations in the student’s answer, sometimes with different LEs. Conversely, a LE in the answer can be matched when searching for different concepts of the ideal answer. For example, the token “and” in the student’s answer is ambiguous as it can mean *AndThen* or *OrThen* in the ideal answer. The SN is used for disambiguation, as explained in the next phase.

**C. Creating the Student’s Semantic Network**

The third phase links the annotated concepts (or relation-, attribute-, literal-concepts) of the student’s answer to nodes of the ideal answer’s SN. The result is the specific SN of the student’s answer, which is an overlay of the ideal answer’s SN. For example, consider the following sentence given by a student: “If *W* is closer than its parent *V*, the parent of *W* is modified to *V*.” Fig. 6 shows a subset of the SN, corresponding to the phrase “parent of *W*”. Table III presents the annotations of the student’s answer. The columns respectively indicate: the SN’s concepts, the corresponding LEs and their positions in the text. Note that the concept *ParentOfW* was not found in the student’s text and hence is marked as absent. However, other concepts, such as *Parent* and *W*, were found more than once, and in
different forms. This is an example of where a given concept can be found in multiple locations in the student’s answer.

The third phase is performed by traversing the SN to find the parameters of every concept. We define the parameters of a given concept as the n-tuple including the concept itself and its children. Every parameter has zero or many corresponding LEs. Fig. 6 shows the parameters of the concept Dereference, which are (Dereference, Parent, W), corresponding to lines 2 to 4 of Table III. To find the parameters, our system relies on the relative positions of the LEs found in the student’s text. It considers the distances between the candidate LEs of each parameter and selects those that optimize the textual proximity, i.e. that minimize the span of their corresponding positions.

We define the span as the sum of the pair-wise distances among the candidate LEs. For example, the parameter Parent is matched to two candidate LEs, “its parent” and “the parent”, at positions 6–7 and 9–10, respectively. The system has to choose one of them. The parameter Dereference, corresponding to “of”, has position 11. The parameter W is found at positions 2 and 12. The system chooses the sequence 9–10, 11, 12, because it has a zero span (the three LEs are contiguous). Therefore, it identifies “the parent”, “of”, and “W” as the LEs of the parameters of the Dereference node. In addition, the system stores in this node the calculated span (zero), the first and last found positions (9–12), and also the corresponding LE (“the parent of W”). These selected positions are indicated as subscripts in Fig. 6.

The system then continues up the semantic graph, identifying the parameters of the other concepts and linking them to the corresponding expressions, as seen before. This implies that recognized concepts become parameters of other concepts. For example, the parameters of the ParentOfW concept are (ParentOfW, Dereference). However, ParentOfW is absent in the student’s text and only one candidate LE remains. Therefore, the system copies the information stored in the Dereference node into the ParentOfW node, which correctly inherits the same LE (“the parent of W”).

While traversing the SN, the algorithm binds concept variables as it discovers them in the student’s answer. The system maintains a symbols table to keep track of the actual names used for the expected variables. If the student uses “X” and “Y” instead of the expected “V” and “W”, the system normalizes it back to “V” and “W”, in the case of Dijkstra’s algorithm. When a student uses a relative pronoun, the referent is supposed to be the closest variable mentioned.

In general, the parameters have no specific order; therefore, the order of the LEs in the student answer is not taken into account. For example, note that W, Dereference, and Parent, cited in Fig. 6, could appear in any order in the student text, such as “parent of W” or “W has parent”. An ordering is only required with the relation-concept AndThen, where the parameter identified by the relation doBefore should come before the parameter identified by the relation doAfter, or with asymmetrical operators.

Note that this semantic representation is language-independent. It does not rely on grammatical patterns. In fact, in our implementation, the course ontology is in English and the student texts and LEs are in French. It is conceivable to adapt it easily to other languages by providing the LEs (and dictionaries) for those languages and linking them to the course ontology. This is possible thanks to the uniform representation of relations, attributes, and literals as concepts, and to a very granular course ontology that maximizes concepts recognition in the texts.

Finally, the algorithm also infers “equals true” for all LEs of type “V is visited”.

D. Grading Procedural Knowledge

The final phase consists of traversing the SN to calculate the similarity measures used by the grading algorithm. Our system assesses knowledge by five similarity measures: part-whole similarity (PS), textual proximity (TP), lexical similarity (LS), ontological proximity (OP), and temporal ordering (TO). All measures are in the $[0,1]$ interval. PS recursively measures the degree of detail given by the student in relation to the expected level of detail given by the teacher. TP measures the distance between related concepts in the student’s answer by considering their position in the text. OP measures the distance between the expected concepts of the ideal answer and the found, inferred, and fuzzily matched concepts, using the course ontology. LS measures the distance between the LEs used by the student and the previously found LEs. Finally, TO compares the ordering of LEs in the student’s answer to the temporal ordering of the ideal answer.

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Found linguistic expressions</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentOfW</td>
<td>absent</td>
<td>—</td>
</tr>
<tr>
<td>Parent</td>
<td>“its parent”, “the parent”</td>
<td>6–7, 9–10</td>
</tr>
<tr>
<td>Dereference</td>
<td>“of”</td>
<td>11</td>
</tr>
<tr>
<td>W</td>
<td>“W”, “W”</td>
<td>2, 12</td>
</tr>
<tr>
<td>V</td>
<td>“V”, “V”</td>
<td>8, 16</td>
</tr>
<tr>
<td>If</td>
<td>“If”</td>
<td>1</td>
</tr>
<tr>
<td>LessThan</td>
<td>“is closer than”</td>
<td>3–5</td>
</tr>
<tr>
<td>Assignment</td>
<td>“is modified to”</td>
<td>13–15</td>
</tr>
</tbody>
</table>
Teacher-provided weights allow adjusting the measures’ contribution towards the students’ grades, according to the teacher’s preferences. In our case, we empirically selected a combination of these weights that gives grades close to those given by human graders with a very strong correlation. This combination showed better results than equal weighting for all the measures. We used the same combination for all students’ answers to all questions. The process of finding the best combination could be automated if desired.

1) Part-Whole Similarity: A matched leaf in the student’s SN scores 1.0, and an unmatched leaf (marked absent in phase two) scores zero. For each internal node and the root, the PS measure is the ratio of the number of matched nodes of its subgraph to the total number of nodes therein, as shown in (1), where \( c_i \) is a concept used in the SN.

\[
PS(c_i) = \frac{\sum_{c_j \in \text{children}(c_i)} PS(c_j)}{\vert \text{children}(c_i) \vert}
\]  

For example, consider Fig. 3, where the OrThen concept has two children: ModifyDvOfW, which has been matched; and ModifyParentOfW, which is absent. Therefore, only half of the child nodes have been matched. The PS measure for the ModifyDvOfW node is 1.0 and that of ModifyParentOfW is zero. Therefore, the PS measure of the OrThen node is: \( (1 + 0) / 2 = 0.5 \). We suppose here that the decomposition of a concept into its subconcepts is a kind of isPartOf relation. We further assume that this relation is transitive.

2) Textual Proximity: This measure is inversely proportional to the distance between the parameters of a concept (cf. Section III-C) in a student text. The TP measure is 1.0 when the LEs of the parameters are contiguous in the student text. It is defined as zero if no parameters are found in the student text for concept \( c_i \). Otherwise, it is calculated by (2). Note that parameters separated by unrecognized or irrelevant words increase the span, thus lowering the TP measure.

\[
TP(c_i) = 1 - \frac{\text{avgspan}(c_i)}{\vert \text{tokens} \vert}
\]  

In (2), \( \text{avgspan}(c_i) \) is the average span of all occurrences of concept \( c_i \) in the SN. In order to give a value in the \([0,1]\) interval, the span is normalized by dividing by the number of tokens in the student answer, \( \vert \text{tokens} \vert \).

3) Lexical Similarity: This measure calculates the resemblance between two LEs, \( e_i \) and \( e_j \). The first one is taken from the student answer, and the second from the dictionary (D2) containing LEs associated to concept \( c_i \). If the two are identical, \( LS(e_i, e_j) = 1.0 \). If they share the same radical (e.g. “node” and “nodes”), then \( LS(e_i, e_j) = 0.6 \). If one is a synonym of the other (e.g. “node” and “vertex”), then \( LS(e_i, e_j) = 0.3 \). Otherwise (e.g. “node” and “graph”), \( LS(e_i, e_j) \) is zero. If the LEs contain more than one word, the LS measure is the maximum of the individual LS values. A similar measure uses the Levenshtein distance in OeLE.

4) Ontological Proximity: This measure is introduced to take into account the fact that a concept \( c_i \) in the ideal answer can be fully or fuzzily matched to a concept \( c_j \) in the student answer. The latter could be a sibling, a parent or a child in the course ontology (cf. Section III-B). In case concept \( c_j \) is absent, but some of its children are matched, \( c_j \) is matched by inference. The OP measure calculates the distance between each occurring concept \( c_i \) of the ideal answer with matched concept \( c_j \) of the student answer.

If concepts \( c_i \) and \( c_j \) have no taxonomic parent, according to the OWL subClassOf relation, or if concept \( c_j \) is absent, \( OP(c_i, c_j) \) is zero. If concepts \( c_i \) and \( c_j \) are an exact match, \( OP(c_i, c_j) = 1.0 \). If \( c_j \) was inferred by the presence of its children, \( OP(c_i, c_j) = 0.5 \). Otherwise, the concepts are fuzzily matched and the OP measure is calculated, as in OeLE’s concept proximity measure, by using the class hierarchy defined by the subclass relation:

\[
OP(c_i, c_j) = 1 - \frac{\vert \text{nodes}(c_i, c_j) \vert}{\vert \text{concepts} \vert}.
\]  

In (3), \( \vert \text{nodes}(c_i, c_j) \vert \) is the number of concepts separating \( c_i \) and \( c_j \) through the shortest path in the class hierarchy, and \( \vert \text{concepts} \vert \) is the total number of concepts in the course ontology [10]. A shorter path thus indicates a stronger similarity between the two concepts.

5) Temporal Ordering: This measure takes into account the ordering specified by the AndThen concepts in the ideal answer. The TO measure is calculated by dividing the number of AndThen concepts having the right ordering of LEs in the student answer by the total number of AndThen concepts in the ideal answer.

6) Grading: To grade a student answer, we use (4) below. All the measures mentioned above are calculated for each concept (or relation-, attribute-, literal-concept) \( c_j \) of the student’s SN and are weighted to become the terms of our equation. Note that we have omitted the detailed parameters of the measures in the equation because of space limitations.

\[
G(R) = \frac{\sum_{c_i \in \text{concepts}(R)} \left\{ \frac{ps \times PS + tp \times TP + ls \times LS + op \times OP + to \times TO}{\vert \text{concepts}(R) \vert} \right\}}{\vert \text{concepts}(R) \vert}
\]  

In (4), \( ps \), \( tp \), \( ls \), \( op \), and \( to \) are weights in the \([0,1]\) interval corresponding to the respective measures. The combination of weights used in our experiments, giving a very strong correlation, is: \( ps = 0.16 \), \( tp = 0.16 \), \( ls = 0.06 \), \( op = 0.39 \), and \( to = 0.23 \).

IV. EXPERIMENTAL RESULTS

To evaluate our system, we used questions and answers from three exams on computer algorithms given at Université de Moncton between 2011 and 2013. The three
The course ontology contains 114 concepts and 17 relations, shared by the three questions. The SNs for DIJ, DFS, and BT respectively contain 77, 9, and 32 concepts (nodes) and 80, 9, and 32 relations (arcs). The reuse of ontology concepts facilitates the adoption of new questions. For example, the grading of a question about breadth-first search would not require adding concepts to the ontology. The SN would reflect the new order of node traversal. However, a question about the B+ tree would use those concepts from a binary search tree, but these concepts might not be sufficient to represent the whole SN.

For DIJ, Table IV shows the grades, over 4 points, given by human graders (G1–G7) for each student (S1–S13). The instructors gave average grades between 1.92 and 3.31, with an average of 2.80 and a standard deviation of 0.78.

Table V shows a comparison of the average human grades for each student with those of the system. The system gave an average of 3.13 points (out of 4), with a standard deviation of 1.17. Note that the average grade assigned by the system falls in 50% of cases within a distance of one standard deviation from Avg-Human. The correlation between the average human grades and those of the system is very strong (0.79, with \( p < 0.05 \)), which indicates a very strong positive relationship.

For DFS, Table VI shows a comparison of the average human grades for each student with those of ProcMark system. The system gave an average of 3.13 points (out of 5), with a standard deviation of 1.24. Note that the average grade assigned by the system falls in 50% of cases within a distance of one standard deviation from Avg-Human. The correlation between the average human grades and those of the system is 0.79 (\( p < 0.05 \)), which indicates a very strong positive relationship.

For BT, Table VII shows a comparison of the average human grades for each student with those of the system. The system gave an average of 4.94 points (out of 10), with a standard deviation of 1.17. Note that the average grade assigned by the system falls in 44% of cases within a distance of one standard deviation from Avg-Human. The correlation between the average results of human graders and those of the system is very strong (0.79, with \( p < 0.05 \)).

Overall, the system gives grades with a very strong positive correlation to human grades. We observe that the system performs well on answers of variable length and different granularity levels of the corresponding SN.
V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new system specially designed for the assessment of free-text answers containing procedural knowledge. We evaluated the system using questions from three exams on computer algorithms given at Université de Moncton. A total of 53 students and 7 human graders participated in our experiments. Overall, the system gives numeric grades with a very strong (0.70, 0.79, and 0.79) positive correlation to human grades. The system introduces several novel ideas. First, to allow the representation of procedural knowledge, we implemented the notion of functional concepts that can be nested and reused. We define a course ontology offering general algorithmic knowledge and choose that all knowledge representation be parameterized as concepts, thus allowing variable granularity levels and the possibility of zooming in and out of these levels. Second, we use semantic networks to represent ideal and student answers. This approach allows the grading system to be language-independent by using an overlay of the semantic network rather than the textual answer. Third, we propose three novel similarity measures adapted to procedural knowledge: part-whole similarity, textual proximity, and temporal ordering.

This approach could be used in other domains where procedural knowledge is central to processing the text. Similar methods are applied in [17], [18] to biomedical ontologies, although not for evaluation. Currently the system is easy to use by professors of Computer Science as they are used to graph terminology. We plan on designing a graphical tool to facilitate the edition of semantic networks by non-Computer Science instructors. Also, the system gives only the grade as a feedback to the students. It would be possible to supply a commented semantic network to each student.

ACKNOWLEDGMENT

We thank the Natural Sciences and Engineering Research Council of Canada for partially financing this research, as well as the Canada Foundation for Innovation for financing the infrastructure and the technical support for this work. We also thank the instructors and students who participated in our experiments.

REFERENCES