Automatic creating variation of CS1 assignments and exams

Filipe Dwam Pereira  
Hermínio Júnior  
Federal University of Roraima, Boa Vista, BR  
filipe.dwan@ufrr.br  

Aileen Benedict  
Mohsen Dorodchi  
University of North Carolina at Charlotte, Charlotte, US  

Elaine H. T. Oliveira  
Leandro S. G. Carvalho  
David B. F. Oliveira  
Federal University of Amazonas, Manaus, BR

The adoption of Online Judge (OJ) environments by CS1 instructors has increased over the last few years [8–11, 14, 17, 19, 22, 24, 27–31]. OJs reduce instructors’ workload in correcting learners’ codes and provide instantaneous and accurate feedback to students about the correctness of their solutions [3, 6, 16, 21, 23, 26, 28]. Despite the benefits, there are still repetitive and laborious tasks to feed OJ systems. For example, the literature [1, 5, 13, 16, 18] recommends that instructors create variations of assignments and exams for different CS1 classes during the semesters to hamper plagiarism practice. By creating variations of assignments and exams, it is more difficult for students to use code solutions from past courses [13]. Indeed, an even more rigorous way of avoiding plagiarism would be to create personalized assignments and exams proactively for each student [5]. However, doing this manually is impractical, especially in classes with a high number of students.

To address this, we intend to create a mechanism for automatically selecting problems to compose new assignments and exams so that the new selection of problems is similar enough to the old in terms of problem topics and challenge levels. Ordinarily, the questions from an assignment available in CS1 courses share the same topic (e.g., conditional structure) [4, 12], and are scaffolded from easier to more challenging problems [7, 15]. In this work, we propose a way to generate N new assignments based on a previous one, called the “master assignment.” These new assignments will then be composed of problems unique to the master assignment but similar in terms of topics and difficulty levels. Additionally, the same reasoning must be used to create new exams.

To accomplish our goal, we propose the procedure illustrated in Algorithm 1. In the procedure getNewList, L = \{q_1,...,q_m\} represents a given master assignment, where m is the number of questions in L. The output L’ = \{q’_1,...,q’_m\} depicts a new assignment which has the same topic of L and requires a resolution effort (i.e., challenge level) similar to that of L. To create more than one new L’s, we can manipulate the global variable K. For example, to create a second L’ using a given L as input, we just need to assign 2 to the global variable K (K ← 2 in line 1 of the algorithm).

Notice that our procedure uses two auxiliary functions: getTopic and findKthNearestNeighbour. In the getTopic function, each question from the OJ we will have the tuple (q, p), where q is the statement of the problem and p is a vector that represents the effort required to solve the question q. Each pair (q, p) has a topic t associated. The possible topics of the questions is based on the CS1 curriculum: Sequential, Composite conditional structures, Chained conditional structures, Repeating structures by count, Vectors and Strings and Matrices. As the questions of many OJs are not annotated with the topic of the question, the function getTopic uses machine learning and natural language processing techniques to predict the topic t of the statement q. More specifically, we will use a word embedding layer representation of each question q in a deep learning model, similar to what we have done in these works [2, 12, 25].

Algorithm 1 Creating new assignment/exam

```
1: global const K ← 1  \* K sets the ith L’ created based on L.
2: procedure getNewList(L)
3:   L’ ← {}  
4:   for (q, p) ∈ L do
5:     t ← getTopic(q)  
6:     k ← K  \* Kth nearest neighbour of p is first used as p’
7:     q’ ← findKthNearestNeighbour(p, t, k)
8:   while q’ ∈ L’ do
9:     k ← k + 1  
10:    q’ ← findKthNearestNeighbour(p, t, k)
11:   end while
12:   L’ ← q’ ∪ L’  \* new assignment/exam L’
13: end for
14: return L’  
15: end procedure
```

To find a problem that requires similar effort, findKthNearestNeighbour is used. Here, we use the nearest neighbour technique over the features, further discussed in previous works where we proposed and validated features to measure the students’ required effort per problem [20, 23, 24]. The features will be the dimensions of the vector p that represents the effort required to solve q. In total, there are 21 features. Given a pair (q, p), the vector p has the aggregation of the features’ values based on the learners who solved that question q. To illustrate, given a question qa, there is a feature called loc which is the lines of code a student used in their solution for question qa. Thus, one of the dimensions of the vector p, will be the average locqa, for all students who submitted accepted solutions for qa.

Finally, we can assume that the questions from L’ is sorted by difficulty level. The reason is that L’ is created based on the master list L, which has been previously sorted by difficult level by an instructor. In line 12 of Algorithm 1, the question q’ in inserted in L’ in the same interaction of the for loop (line 4) when q ∈ L is accessed. That is, as the pair of questions (q, q’) are potentially from the same topic and requires a similar effort to be solved, hence, L and L’ are arranged in the same order.
ACKNOWLEDGMENTS
This study, carried out within the scope of the Samsung-UFAM Project for Education and Research (SUPER), according to Article 48 of Decree n° 6.008/2006 (SUFrama), was partially funded by Samsung Electronics of Amazonia Ltda., under the terms of Federal Law n° 8.387/1991, through agreements 001/2020 and 003/2019, signed with Federal University of Amazonas and FAEPI, Brazil. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq grant 308513/2020-7).

REFERENCES