Design of an Automated Pseudo-Code Grading Tool

Patrick McDowell, Christine Terranova, Kuo-Pao Yang
Department of Computer Science
Southeastern Louisiana University
Hammond, LA 70402 USA

Abstract— **The intent of this paper is to present a method for the automated grading of language-independent code segments in computer science classes. The proposed grading tool focuses on the overall structure and logic of the code segment rather than on the syntax or formatting constraints that exist within specific programming languages. For that purpose, when a student inputs code, that code is first processed into a standardized pseudo-code before analysis of the code is performed and a grade is outputted. Currently, a sliding correlation is used to measure the accuracy of the inputted code compared to an "answer key" (key code). The correlation is performed on the entire block of code using condition statements and loops as peaks in the processing phase. The output of the grading tool will yield a value between 0 and 1 that can be converted to a point or percentage value depending on an instructor's preference.**

Keywords- Automated grading, pseudo code analysis

I. INTRODUCTION

Although enrollment in computer science courses has been on the rise, the number of faculty positions in the discipline has not increased in proportion [1]. To accommodate for this, many faculty are forced to expand their work loads and to handle larger class sizes more often than not [2]. Increasing the capacity of each course and then having to teach more of those courses have encouraged some faculty to search for methods that will lessen grading time while still providing thorough assessments.

Many online grading systems are comfortable handling true-false, multiple-choice, and matching type problems. Those systems also typically allow for short-answer or fill-in-the-blank type problems where the answer is either based on a numeric computation and comparison, or an accurate match to some element in a subset of correct choices. All of the aforementioned question formats can be automatically graded within the online grading system which can alleviate the grading time for assignments that make use of those question types. However, an essay question type is the most applicable format that common online grading systems would allow for code input. Yet an essay question must still be graded by hand.

For faculty wishing to create assessments that are a combination of auto-graded question types with a few coding (essay type) questions, then current online grading systems may be acceptable because the amount of grading can be greatly reduced. On the other hand, faculty whose assessments would be watered down by the standard auto-graded question types cannot benefit from having the majority of their questions created as essay types since they would still need to hand-grade those problems. Although using a combination of both auto-graded and hand-graded questions is one solution to lessening grading time, the proposed automated pseudo-code grading tool can be applied to hand-graded questions regarding programming concepts by producing a score based on the structure and logic behind code segments regardless of syntax or the programming language used.

II. BACKGROUND

The automatic grading of code is not unheard of, and there are tools that exist that make the task possible. The free, open-source platform INGInious directs its attention towards MOOCS (massive open online courses) by creating scalable, virtual enviroments. Once it is installed and configured, courses, exercises and tasks can be created where code can be securely executed, checked, and then feedback is provided. In order to execute the code, container images are available that allow for the use of different languages. INGInious is compatible with some learning management systems such as edX and Moodle [3].

Web-CAT is a common, widely-known grading tool developed at Virginia Tech that puts emphasis on test-driven development (TDD). Using this system, students are graded on how well they test their own code which inspires critical thinking, engagement in assignments, and attention to detail. Web-CAT is most often used to grade JAVA and C++ source code using TDD tools such as JUnit and XUnit [4, 5].

MIT CSAIL has also produced a system to assist with grading programming assignments. Their platform provides students with feedback on programming exercises written in Python. Error models are utilized in order to generate feedback that is based on the line number of where an error occurs, the expression that could be causing the error, and a possible modification of that expression that could fix the error. The tool checks equivalences on the program with substantially more inputs than the typical test-case technique, but the feedback suggestions are based on low-level functional associations [6].

All of the above tools require some specified programming language environment to be able to automatically grade source code. While that may be useful in introductory programming classes where syntax and error handling are being taught along with good coding practices, some courses place more focus on the logic behind the code. In those courses, pseudo-code is typically used to write algorithms since it provides a way for students to prove that they understand a given task by illustrating a logical approach without the need of programming language constraints. In fact, pseudo-code is such a fundamental aid in understanding source code that the

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 8 Issue 12, December-2019

Nara Institute of Science and Technology has developed a tool called Pseudogen that generates pseudo-code from Python source code using tree-to-string machine translation which yields a natural language interpretation for each line in the code. Currently, the system does not grade the inputted code but rather is used as a learning tool [7].

III. APPROACH

The idea behind the proposed automated grading system is that the tool should compare a student's inputted code response to a set of correct answers, and then output a grade for the inputted code based on how structurally and logically correct the code is. Again, this grading is accomplished independent of a programming language. The inputted code does not need to be compiled or executed for the grading algorithm to be performed, and it does not need to follow syntax constraints.

McDowell illustrates how the student's inputted code can be converted into a signal. Each loop that is entered yields an increment of 1000 in height whereas each conditional statement yields an increment of 200. Preprocessing of the inputted code alleviates noise and attempts to make the computation of the sliding correlation as accurate as possible. During the preprocessing phase, comments, excess blanks, new lines, and tabs are removed. Also, print statements and strings are reduced for simplicity. Afterwards, variables are located in the code and assigned a value of v_i where i is a count of the number of variables found in the code. Variables in the answer key are then best matched to variables in the student code and the indices are changed accordingly.

Once the preprocessing has been accomplished, each character in the code is converted to its ascii integer value and stored in an array (this is done for the answer key and for the student's inputted code). The offset for the loops and condition statements are added, and the sliding correlation is performed. The maximum value obtained from the sliding correlation is outputted [5 McDowell].

The primary contribution of this work is the standardization of the inputted pseudo code shown in Fig. 1. The following paragraphs provide detail.

Before sending the code through the sliding correlation, the code is standardized and preprocessed for the most accurate structural comparison. For example, the system is not intended to grade code based on whether or not a for loop is used instead of a while loop. Thus, the system converts all loops to a while-loop format and replaces the keyword for the loop with the word “loop:” followed by the condition for exiting the loop. All condition statements are treated similarly. See the following examples:

INPUTTED PSEUDO CODE

```
1  int x = 0;
2
3  for(int i = 0; i < 10; i++) {
4     if (x > 0) {x = x – 2*i}
5     else {x = x + 2*i}
6  }
```

PSEUDO CODE STANDARDIZED

```
1  x = 0
2  i = 0
3  loop: i < 10
4    {
5    condition: x > 0
6          {
7          x = x – 2*i
8          }
9    condition: x <= 0
10         {
11         x = x + 2*i
12         }
13   i++
14   }
```

Fig. 1. The code first snippet of code represents a student’s inputted code, and the second is the pseudo-code generated.

Note that students must either use braces or proper indentation to signify loop and conditional blocks. The standardization should do the following:

- Convert all letters to lowercase
- Eliminate data type declarations
- Remove end syntax and empty lines
- Change “do-while”, “for”, and “while” loops into a formatted “loop:” equivalent
- Change “if-else” statement types into formatted “conditional:” equivalents
- Represent the negated condition statements for “else”
- Indent each block of code based on depth of nesting
- Use braces to denote the beginning and ending of code blocks

Recall that once the pseudo-code has been generated, the preprocessing phase of the current system will enhance the comparison of the code to an answer key by:

- Removing comments, excess blanks, new lines and tabs
- Simplifying print and string statements
- Locating and assigning a subscript count to each variable in the code
- Matching the variables in the answer key to the most adequate variable used in the student code
- Converting characters to an ascii integer value
- Adding offsets depending on whether a loop or condition has been applied

IV. RESULTS AND CONCLUSIONS

The following paragraphs show a typical example of the process and compare it the results before the pseudo code standardization process was developed.

Fig. 2 below illustrates how the pseudo-code standardization improves the grading of the overall code structure, suppose that the correct answer for the example shown in Fig. 1 is given to be:

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

```
1  number = 0
2  n = 10
3  for(i = 0; i < n; i++) {
4     if (i%2 == 0)
5           {number = 2*i}
6      else
7           {number = -2*i}
8     }
```

```
1  number = 0
2  n = 10
3  i = 0
4  loop: i < n
5     {
6     condition: i%2 = 0
7           {
8           number = 2*i
9           }
10    condition: i%2 != 0
11          {
12          number = -2*i
13          }
14    i++
15    }
```

Fig. 2. The first code snippet represents the instructor's answer key code, and the next is the pseudo-code generated.

Prior to the pseudo-code standardization, the sliding correlation yields a value of 0.7523. However, after the code has been put through the pseudo-code standardization, the sliding correlation produces a result of 0.9535. As far as the overall structure is concerned, there is one loop and two condition statements nested within it. The student's code is structurally similar to the key code which is why a strong correlation exists. As illustrated in Fig. 3 and Fig. 4, the structures look more similar after the pseudo-code standardization has been performed.

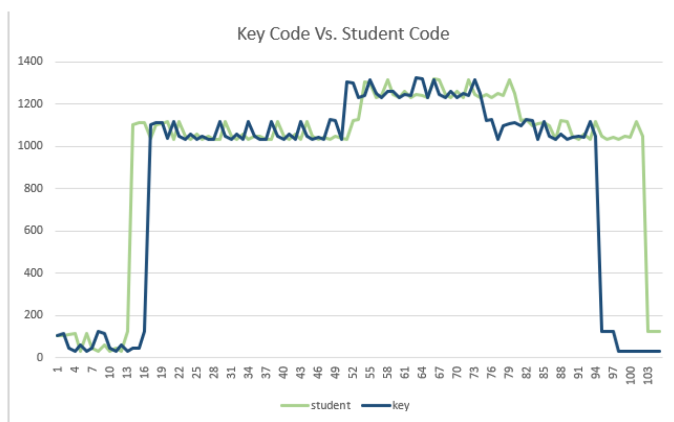

Fig. 3. The integer offset values for the original key code and original inputted student code.

Upon further investigation, it can be seen that the student's code is not logically equivalent to the key code. For the student, the primary variable x results in {0, 0, 2, -2, 4, -4, 6, -6, 8, -8} whereas in the key the primary variable number results in {0, 2, -4, 6, -8, 10, -12, 14, -16, 18}. Thus, even though the overall structure of the program has been appropriately graded, the logical details of the code need to be refined.

V. FUTURE GOALS

Although the pseudo-code standardization has improved the grading of the overall expected structure of inputted code, an adequate measurement of logic correctness must also exist in order for the proposed automated grading system to output accurate results. To assess the logic in the code, there are a few techniques that may be applied:

1. Use the given exiting conditions for the loop and condition statements to create vectors that will alter the shape of the "signals" produced in the code rather than just using offsets of 1000 for loops and 200 for condition statements.
2. Ascertain if each line in the key code matches an equivalent line in the inputted code.
3. Calculate a sliding correlation for each matching block of code within a loop or condition statement.
4. After locating all variables within the code, use instance values for those variables to determine if the correct computations are being applied.

Through small-scale testing it can be determined if all or some of the above techniques should be used. However, once satisfied with a measurement of logic analysis, an overall grade can be given on the inputted code by applying weights to the score of structural adequacy and the score of logic correctness.

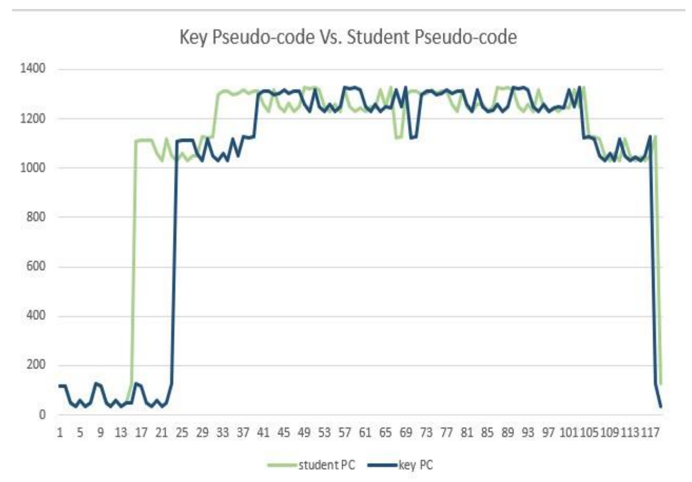

Fig. 4. The integer offset values for the key code and inputted student code after the pseudo-code standardization has been applied.

REFERENCES

[1] T. Camp, R. Adrion, B. Bizot, S. Davidson, M. Hall, S. Hambrusch, E. Walker, and S. Zweben, "Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006," Computing Research Association, February, 2017, retrieved from https://cra.org/wp-content/uploads/2017/02/Generation-CS.pdf.

[2] E. Roberts, "A History of Capacity Challenges in Computer Science," March, 2016, retrieved from http://cs.stanford.edu/people/eroberts/CSCapacity.pdf.

[3] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. Roy, "Automatic Grading of Programming Exercises in a MOOC Using the INGInious platform," Experience Track Proceedings of the European MOOC

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

Stakeholder Summit 2015 (EMOOCs 2015), pp. 86 – 91, 2015, retrieved from https://www.info.ucl.ac.be/~pvr/DervalEMOOCS2015.pdf.

[4] S. Edwards, “Automatically Grading Programming Assignments with Web-CAT,” June, 2013, retrieved from http://web-cat.org/cta13/cta13-Web-CAT-tutorial.pdf.

[5] P. McDowell, “Automated Logic Analysis Using a Sliding Correlation,” Journal of Emerging Trends in Computing and Information Sciences, vol. 8, no. 1, pp. 17 – 20, January 2017.

[6] R. Singh, S. Gulwani, A. Solar-Lezama, “Automated Feedback Generation for Introductory Programming Assignments,” Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13), pp. 15 – 26, June, 2013.

[7] H. Fudaba, Y. Oda, K. Akabe, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura, “Pseudogen: A Tool to Automatically Generate Pseudo-code from Source Code,” Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15), pp. 824 – 829, November, 2015.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)