Difference Syntax Trees for Characterising Student in Programming Course

1st Kouta Aoki
Department of Advanced Information Engineering.
National Institute of Technology (KOSEN), Nara College
Nara, Japan
ai1139@nara.kosen-ac.jp

2nd Hidetake Uwano
Department of Information Engineering.
National Institute of Technology (KOSEN), Nara College
Nara, Japan
uwano@info.nara-k.ac.jp

***Abstract*—The Online Judge System (OJS) is well-used in programming courses at universities or for self-learning. The system compiles and executes a set of source codes submitted in response to an assignment and automatically grades them by comparing the output results and expectations. In university programming courses, especially courses for beginners, students repeatedly modify and submit source code until they receive a 100-point score. The OJS stores every source code until each student gets 100 points on an assignment; the differences through the first to last submissions contain helpful information to estimate students' understanding of syntax or learning units in the course. In this study, the authors propose difference flow, a series of syntax trees extracted from the differences between the final submission and every previous one. The difference flow contains the node where the difference with the 100-point source code and each parent node; hence, its features (such as the count of each syntax node throughout the flow) may indicate the students' understanding.**

***Index Terms*—Programming education, Difference flow, Syntax tree**

I. Introduction

Online Judge System (OJS) is a well-used system that automatically grades submitted source code. The OJS is used in self-learning and programming courses at universities, and some research applies the OJS to programming education [1] [2]. In an educational use, a student submits a source code that corresponds to the assignments provided by the teachers. The OJS compiles and executes the submitted source code and provides feedback, such as correct/incorrect test cases, scores, and compile/run-time errors. Based on the feedback, students repeatedly modify and submit the source code, aiming to implement a 100 (full marks) source code.

OJS stores every source code until each student gets 100 points on an assignment. In this study, the authors focus on the source code differences between versions from a student. Several studies analyze the characteristics of editing behavior based on source code differences [3] [4]. Some beginners in programming courses repeatedly modify the source code for a single problem without understanding the cause when the code works differently from the expected behavior. The differences until the student gets 100 scores are helpful information for identifying a lack of understanding of a specific syntax element or learning unit.

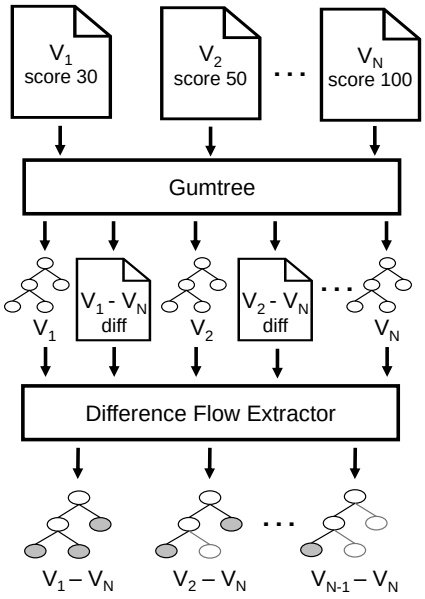

Fig. 1. Our proposal tool

In this paper, the authors propose a tool to automatically extracts the differences between syntax trees parsed from the source codes which the student takes 100 scores and all previous for one assignment. The difference in syntax trees expresses a single edit as edits to different granularities, such as arithmetic expressions, blocks, and methods. For example, three continuous differences 1) `i=i+1`, 2) `print(i)`, 3) `i<10` are also expressed as edits to lines belonging to the same `for` block. The syntax tree-based differences indicate that each edit belongs to the same block, method, and class. A frequent syntax element in the sequence of differences between the source codes of 100 scores and all previous indicates a block where the error cannot be corrected during their work, i.e., the student needs any support or review. In this paper, we define a *difference flow* as a different sequence of syntax trees between the final submission and each previous submission.

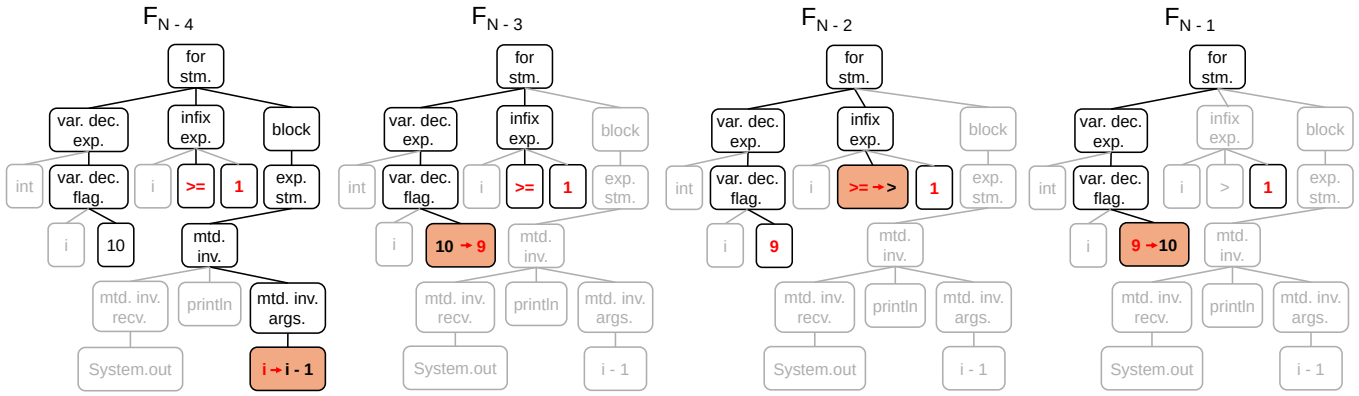

Fig. 2. An example of difference flow

II. Difference Flow

Figure 1 shows the output process of the difference flow. Input every version of the source code for the same assignment from one student, Gumtree [5] outputs 1) the syntax tree for each version, and 2) the diff between the final submission (V_N) and each of the previous versions ($V_1 \ldots V_{N-1}$). We use type (add, delete, move, modify), row, line, and text from the diff information. The *difference flow extractor* integrates them and outputs difference flow. Figure 2 shows a part of the difference flow as an example. F_x is the difference syntax tree of V_x and V_N, while F_{N-1} shows a last difference. The red text shows a difference between V_N, the orange node shows a difference from the previous version, and the gray node shows there is no change until V_N.

The example shows that 1) a series of edits are concentrated to the same `for` block, and 2) the student seeks a correct set of initialize, condition, and loop statements. The differential flow indicates that the student does not fully understand the relationship between the conditional expression of the for statement and the number of loops. Based on this information, teachers can provide additional instruction to the student focusing on the conditional expressions and the loop number. Compared with ordinal text diff from continuing two versions, the difference flow contains parent nodes of each changed node. Hence, the teacher to easily understand that the series of edits is concentrated in the same block, even if the changed lines are located in distant places. It is noteworthy that the difference flow is represented in XML format; OJS can systematically analyze how each student's edit is concentrated on each syntax element, such as counting the number of each syntax element in the difference flow. Using the analysis result, OJS can provide appropriate support for the student, such as suggestions for supplemental assignments about `for` statemetns.

III. Future Work

The difference flow shows the process of a student seeking their full mark score source code for an assignment. The future work with the difference flow is as follows.

- Extraction of student's stuck
  The difference flow contains multiple changes for the same place in the source code until the full score is obtained or until non-corrected errors across multiple source code versions.
  In the example shown in Figure 2, the number of VariableDeclarationFragment (initialization of index variable) and InfixExpression (loop condition) in the `for` statement was high, indicating that the student was stuck specifying the appropriate index value corresponding to the required process. By parsing the difference flow's XML and counting the frequency of node occurrences, it is possible to detect a student's stuck syntax elements and/or specific blocks.
- Scoring understanding of learning unit
  Difference flow of multiple assignments from a single student enables automatic scoring of understanding for learning units (such as syntax elements) in programming courses. Syntax-wise integration of multiple difference flows enables a more accurate assessment of particular syntax elements, such as else-if ladder, without well-designed Unit Tests.

References

[1] H. Sun, B. Li, and M. Jiao, "YOJ: An online judge system designed for programming courses," In Proceedings of the 9th International Conference on Computer Science Education (ICCSE2014), pp.812–816, 2014

[2] W. Zhou, Y. Pan, Y. Zhou, and G. Sun, "The framework of a new online judge system for programming education," In Proceedings of ACM Turing Celebration Conference (TURC2018), pp.9–14, 2018

[3] A. Koyuncu, K. Liu, T. F. Bissyandé, D. Kim, J. Klein, M. Martin, and Y. L. Traon, "FixMiner: Mining relevant fix patterns for automated program repair," Empirical Software Engineering Vol.25, No.3, pp.1980–2024, 2020

[4] H. A. Nguyen, A. T. Nguyen, T. T. Nguyen, T. N. Nguyen and H. Rajan, "A study of repetitiveness of code changes in software evolution," 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.180–190, 2013

[5] J. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing," In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE2014), pp.313–324, 2014