So, H. J. et al. (Eds.) (2020). Proceedings of the 28th International Conference on Computers in Education. Australia: Asia-Pacific Society for Computers in Education

Experimental Design of Automated Extraction for 3-Level Tutoring Support Information in Programming Exercises

Yasuhiro NOGUCHI[a*], Kousei AYABE[a], Koichi YAMASHITA[b], Satoru KOGURE[a], Raiya YAMAMOTO[c], Tatsuhiro KONISHI[a] & Yukihiro ITOH[d]
[a]*Faculty of Informatics, Shizuoka University, Japan*
[b]*Faculty of Business Administration, Tokoha University, Japan*
[c] *Faculty of Engineering, Sanyo-Onoda City University, Japan*
[d] *Shizuoka University, Japan*
*noguchi@inf.shizuoka.ac.jp

Abstract: In programming classes, teachers and teaching assistants (TAs) cooperate to support learners' programming exercises. In many situations while providing support, however, teachers cannot find and resolve the impasse for individual learners while at the same time monitoring the impasse trend in the entire class to effectively provide learners with additional information. In this paper, we categorized 3-level tutoring support information for teachers' activities and designed the experimental architecture for a system that supports automated extraction for the 3-level information from learners in programming exercise activities.

Keywords: Automatic impasse detection, programming exercise, monitoring, decision support

1. Introduction

In recent years, the development of an information society has required more information engineers, including system engineers. In this context, programming skills are not only necessary for such engineers, but are becoming fundamental skills for most of them to utilize information systems in their fields. As well as science or engineering colleges and technical schools for such engineers, liberal arts colleges and ordinary high schools also provide some programming education classes. In early programming education, the following two teaching methods and their mixed method like PBL are often used to acquire basic programming skills:

- A lecture-style method whereby students learn the theory of programming, syntax of programming languages and behaviors, and features of algorithms by listening to the teacher's lecture and reading textbooks themselves.
- An exercise-style method whereby students learn the practical skills of programming by completing tasks based on the lecture content. While the learners attempt to complete the exercises, the teachers circulate among the learners to answer their questions, check the work of learners who are having difficulties, and to provide learning support.

However, novice learners occasionally experience impasses during coding exercises. They might be unable to recover from an impasse for a long time, and therefore cannot proceed with their exercise. For example, syntax errors are the most common errors for them; as reported by Denny, Luxton-Reilly, David, and Hendrickx (2011) based on their analysis of Java programing exercise classes, "some students were often unable to solve their syntax problems." Becker et al. (2011) summarized various viewpoints for improving compilers' error messages for novice programmers (readability, cognitive load, provide context, show examples, scaffolding, logical argumentation, and so on). It has been shown that various measures are necessary to convey the message to learners. Furthermore, learners often reach an impasse where they cannot construct the steps for implementation, cannot find the location (line number) of the codes that have caused the runtime error, cannot solve the run-time error, cannot complete the procedures to attain the expected output values with the expected behavior of the program, and other problems. The cause of an impasse basically depends on the individual situation. Moreover, learners cannot identify the impasse situation, and even if they do have opportunities to ask teachers, they may not be able to explain the situation correctly.

In programming education, many learners complete the exercises using their own PCs, while a small number of teachers and teaching assistants (hereinafter, “teachers”) support them by circulating to answer their questions. Recent novice programming classes are typically one-to-many teaching situations that may face the following difficulties:

A) Difficulty in finding and resolving the impasse for individual learners
Since the learner completes the exercise using their own PC, it is difficult for the teachers to identify what student has reached an impasse in the exercise. Moreover, when the teachers take the learner’s PC to identify the cause of their impasse, it shortens the time the learner has in which to complete the exercises.
B) Difficulty in monitoring the overall class trend in learners’ impasses
Since individual teachers do not have time to share their support results, it is difficult for them in the one-to-many teaching situation to monitor the overall impasse trend in the class in real time. Thus, in the class, it is difficult for teachers to provide additional instructions to learners based on the common types of impasse and common causes of impasse amongst the class learners. Moreover, they cannot update their teaching materials based on the types of impasse identified.

Regarding the detection of impasses and mistakes, several methods have been proposed to detect these by collecting and analyzing learners’ compilation errors and by analyzing learners’ programming activities such as keyboard, mouse click, et cetera (Mazza & Dimitrova, 2004; Biswas & Sulcer, 2010; Hartmann, MacDougall, Brandt, & Klemmer, 2010). However, although some automated feedback tools focus on error detection, they do not provide sufficient feedback for novice programmers (Keuning, Jeuring, & Heeren, 2008). McCall and Kolling (2019) show that there is a large difference in the frequency of occurrence and difficulty of each type of problem for novice programming learners. Therefore, it is difficult to provide guidelines in advance for the problems that may occur. Furthermore, Brown and Altadmri’s (2014) study showed some discrepancy between the frequency of problems recognized by teachers (like many learners being troubled by a particular problem) and the actual frequency of learners’ problems. It is necessary, therefore, to provide individualized tutoring based on the actual learner’s situation, and to rely less on teachers’ intuition. It is also important to monitor the real-time learners’ impasse trend in the classroom.

To reduce these difficulties, in this research, we classified information supporting the difficulties A) and B) into three levels that can be analyzed with the impasse detector (Yamashita et al., 2017). We suggested an experimental design for the system supporting automated extraction of the 3-level information from learners.

2. 3-Level Tutoring Support Information

In this section, we describe the requirements for the monitoring system to support the role of teachers in the exercises portion of learning programming: circulating among the learners to answer their questions, providing additional instructions for them, and updating teaching materials to eliminate their impasses. To reduce the difficulties A) and B), the following functions are required for the system:

A1) A function that provides teachers with a trigger for circulating to a specific learner by automatically identifying whether the learner is in an impasse or not.

A2) A function that provides teachers with tutoring information regarding the learner who has reached an impasse without suspending the learner's programming activities before the teacher circulates to that learner.

- What is the location of the latest learner's code in their impasse?
- What is the location of the teacher's correct answer code corresponding to the latest learner's code in their impasse?
- What are the causes of the impasse?

B1) A function that identifies the number and tendency for learners to reach an impasse as a trigger for teachers to provide additional instructions and/or to re-explain the exercise to the entire classroom.

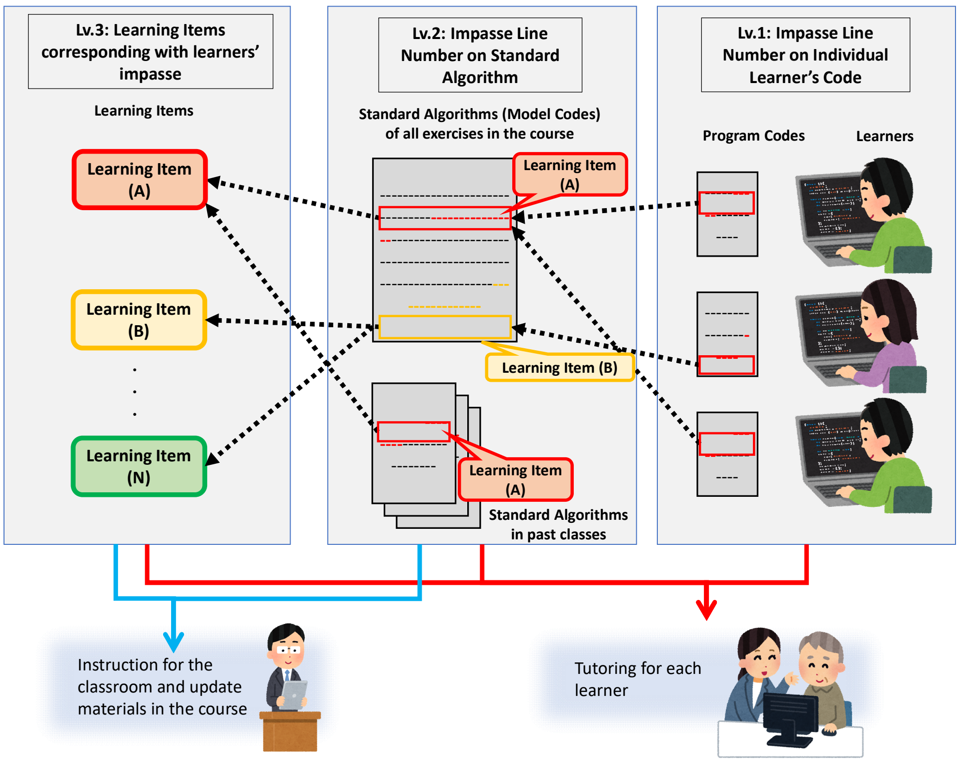
B2) A function that extracts common impasses for learners.

- What are the common locations in the teacher's correct answer code in learners' impasses in the classroom?
- What are the common causes for learners' impasses in the classroom and/or the course of the programming exercise class?

In previous research (Yamashita et al., 2017), the impasse detector focused on detecting some signs of impasse based on individual learners' coding activities in real-time. Teusner, Hille, and Staubitz (2018) also detected some learners' impasses from the tendency of keyboard input (keyboard interventions, and so on). Therefore, these studies could be positioned to contribute to the realization of the function A1). However, to realize the function A2), the system should identify the location (line number) in the learner's code at which the learner has reached an impasse. Additionally, the learner's impasse location in their code, identified in previous monitoring time, should be traced to the learner's latest code, if the location is moved because the learner has edited the code for another purpose. Moreover, the location should correspond to the location of the teacher's correct answer code in order to understand the learner's code in the context of the current programming exercise problems. Furthermore, providing some candidates regarding why the learner has reached an impasse is useful for teachers when preparing their support plan for learners.

Regarding B), it is difficult for a small number of teachers to aggregate individual impasse conditions for many learners. Thus, the monitoring system should support teachers by summarizing and extracting the situations that occur in the classroom. To realize the function B1), not only should the system identify the occurrence of a learner's impasse, but it should also identify whether the sign of impasse is continuing or not, and when the sign disappears. To realize the function B2), the system should aggregate learners' codes, and analyze them for correspondences based on the teacher's model code since each learner's code is different. If the system aggregates learners' common impasses in the classroom on the teacher's model code, the teacher can then prepare additional instructions and identify points of update for the teaching materials on that basis. Additionally, the common causes of learners' impasses could be aggregated not only as a class, but also whole classes in the course can be used to update teaching materials and scaffolding strategies.

Based on the above, we propose a framework of automated extraction for 3-level tutoring information as shown in Figure 1. Lv.1 is the impasse line number of the location on each learner's latest code. Lv.2 is the impasse line number of the location on the teacher's model code corresponding to each learner's impasse. Furthermore, Lv.2 information shows the impasses common to multiple learners. Lv.3 comprises the learning items that allow learners to resolve their impasses corresponding to each learner's impasse. The Lv.3 information is aggregated to identify learning items that are common to learners in multiple classes.



Lv.3: Learning Items corresponding with learners' impasse
Learning Items
Learning Item (A)
Learning Item (B)
Learning Item (N)
Lv.2: Impasse Line Number on Standard Algorithm
Standard Algorithms (Model Codes) of all exercises in the course
Learning Item (A)
Learning Item (B)
Learning Item (A)
Standard Algorithms in past classes
Lv.1: Impasse Line Number on Individual Learner's Code
Program Codes
Learners
Instruction for the classroom and update materials in the course
Tutoring for each learner

Figure 1. 3-Level Tutoring Information in the Framework

3. Experimental Design for Proposed System

3.1 Architecture of the System

Figure 2 shows the architecture of the system. The Learner's Activity Collector collects the learner's codes, compiles a log, and provides the execution log of the successfully compiled program whenever the learner compiles their codes. These data are then sent to Impasse Analyzer Lv.1, Lv.2, Lv.3 in sequence, and these analyzers extract Lv.1, Lv.2, Lv.3 information. The teacher creates some model codes (hereinafter, "standard algorithm") in advance and adds links on certain lines of the model codes to learning items in the learning materials used in the programming course.

3.2 Impasse Analyzer Lv.1

While the existing impasse detector (Yamashita et al., 2017) only identifies whether the learner has reached an impasse or not, our impasse detector observes learners' coding activities and can observe a learner's focused location by presenting a sign of the learner's impasse based on 10 rules in the impasse detector. Thus, for the prototype implementations for the system, a line number is provided when the sign of impasse appears. Additionally, by analyzing the changes from the immediately preceding codes each time, the record in the latest learner's code succeeds the impasse history at the corresponding location. Finally, the analyzer recodes the impasse types and their accumulated number on the line in the latest learner's codes, which enables them to continuously measure learners' impasses at the same location.

3.3 Impasse Analyzer Lv.2

The analyzer records impasse types and their accumulated number and the number of learners who reach the same impasse on the line in the standard algorithm. This is realized by analyzing learners' codes and applying a standard algorithm to synchronize corresponding lines based on their code design structures. In many studies, static analysis for the program code is used to create a correspondence between each learner's codes and the teacher's model codes. It is known that if the constraints on the learners' developing codes are strict in the exercise portion of the class, static analysis can be performed for line-by-line correspondence between the learner's code and teacher's model code (Keuing, Jeuring & Heeren 2018). However, if the constraints of the learners' code are strict, this limits the possible exercises that teachers can design, particularly for developing learners' practical skills.

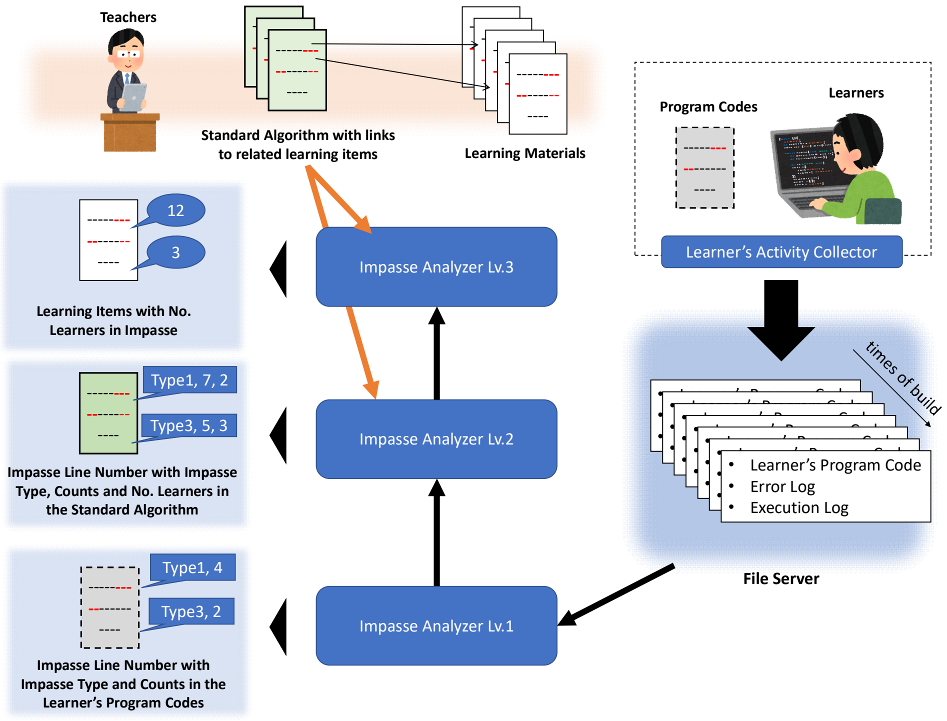

Teachers
Standard Algorithm with links to related learning items
Learning Materials
Program Codes
Learners
Learner's Activity Collector
12
3
Learning Items with No. Learners in Impasse
Impasse Analyzer Lv.3
Type1, 7, 2
Type3, 5, 3
Impasse Line Number with Impasse Type, Counts and No. Learners in the Standard Algorithm
Impasse Analyzer Lv.2
times of build
Learner's Program Code
Error Log
Execution Log
File Server
Type1, 4
Type3, 2
Impasse Line Number with Impasse Type and Counts in the Learner's Program Codes
Impasse Analyzer Lv.1

Figure 2. Architecture of the System (analysis flow example for one learner)

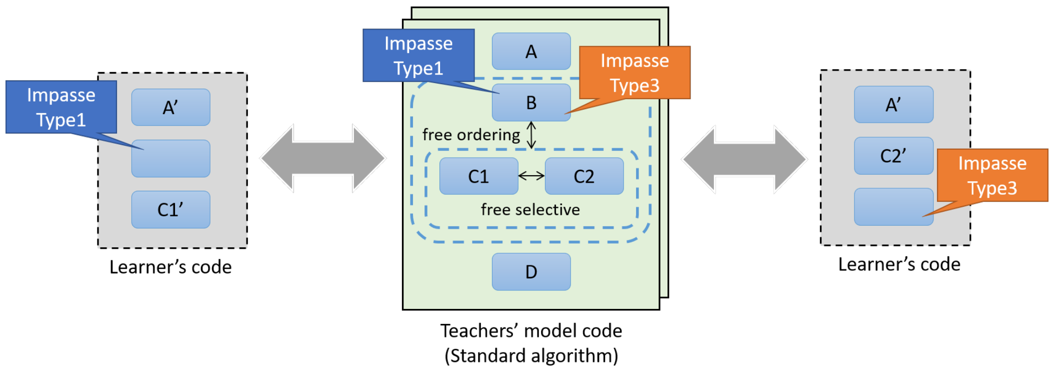


Figure 3. Comparison between learners' codes and standard algorithm in Impasse analysis L.v.2

Therefore, in this paper, we used the format of "standard algorithm" in Konishi, Suzuki, and Itoh (2000) to allow some variations in the teacher's model code. The standard algorithm is described in extended PAD (PAD is a structured diagram developed by (Futamura, Kawai, Horikoshi & Tsutsumi (1980)), and enables teachers to write model codes that include variations in the ordered blocks and selective blocks in Figure 3. Most implementations for algorithms in programming classes for novice learners include free ordering blocks (where the same results are executed from proceeding block A and B as in proceeding B and A), and free selective blocks (where the same results are executed from proceeding A and proceeding B). By this means, multiple patterns of model code variations can be prepared relatively easily.

3.4 Impasse Analyzer Lv.3

When teachers create a standard algorithm, they add links from lines on the standard algorithm to learning items to solve the impasse in the lines. The lines in the standard algorithm record the number of learners who have reached an impasse and the number of times an impasse occurs on a given line. As a result, from the findings regarding which learning items are related with a greater number of impasses, teachers can learn which learning items are likely to bring learners to an impasse in the classroom due to lack of understanding.

4. Conclusion

In this research, to reduce teachers' difficulties when circulating amongst learners during programming exercises, we designed a framework for collecting 3-level tutoring information from learners completing the exercise. We proposed a prototype system to extract this information based on an existing learners' impasse detector and static analysis between learners' codes and teachers' model codes. In the future works, to increase the performance of the function for problem A) and B), it is necessary to tune the threshold function and clarify the restrictions/expectations for learners while adjusting to real programming exercise situations.

Acknowledgements

This study was supported by Japanese Grants-in-Aid for C19K122650.

References

Becker, B., Denny, P., Pettit, R., … Prather, J. (2019). Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *ITiCSE '19: Innovation and Technology in Computer Science Education* (pp. 177–210). doi:10.1145/3344429.3372508.

Biswas, G., & Sulcer, B. (2010). Visual exploratory data analysis methods to characterize student progress in intelligent learning environments. *2010 International Conference on Technology for Education (T4E)* (pp. 114–121). IEEE.

Brown, N. C., & Altadmri, A. (2014). Investigating novice programming mistakes: Educator beliefs vs. student data. *Proceedings of the Tenth Annual Conference on International Computing Education Research* (pp. 43–50). ACM.

Denny, P., Luxton-Reilly, A., David, I. & Hendrickx, J. (2011). Understanding the syntax barrier for novices. *ITiCSE '11: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (pp. 208–212). https://doi.org/10.1145/1999747.1999807.

Hartmann, B., MacDougall, D., Brandt, J., & Klemmer, S. R. (2010). What would other programmers do? Suggesting solutions to error messages. *Proceedings of the SIGCGI Conference on Human Factors in Computing Systems* (pp. 1019–1028). ACM.

Keuning, H., Jeuring, J., & Heeren, B. (2008). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, No. 3. https://doi.org/10.1145/3231711

Konishi, T., Suyama. A., & Itoh. Y. (1995) Evaluation of Novice Programs Based on Teacher's Intentions, *International Conference on Computers in Education*, (pp.557-566).

Mazza, R., & Dimitrova, V. (2004). Visualizing student tracking data to support instructors in web-based distance education. *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters* (pp. 154–161). ACM.

McCall, D., & Kölling, M., (2019). A new look at novice programmer errors. *ACM Transactions on Computing Education*, July 2019, Article No.38. https://doi.org/10.1145/3335814

Teusner, R., Hille, T., & Staubitz, T. (2018). Effects of automated interventions in programming assignments: Evidence from a field experiment. *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, no.60, pp.1–10. https://doi.org/10.1145/3231644.3231650

Futamura, Y., Kawai, T., Horikoshi, H., & Tsutsumi, M. (1980). Disign and Implementation of Programs by Problem Analysis Diagram (PAD), Journal of Information Processing Society of Japan, Vol.21, No.4, (pp.259-267). (in Japanese)