Rodrigo, M. M. T. et al. (Eds.) (2021). Proceedings of the 29th International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Analysis of the Answering Processes in Split-Paper Testing to Promote Instruction

Shin UENO*, Yuuki TERUI, Ryuichiro IMAMURA, Yasushi KUNO & Hironori EGI
[a]*Department of Informatics, The University of Electro-Communications Tokyo, Japan*
*u2130010@edu.cc.uec.ac.jp

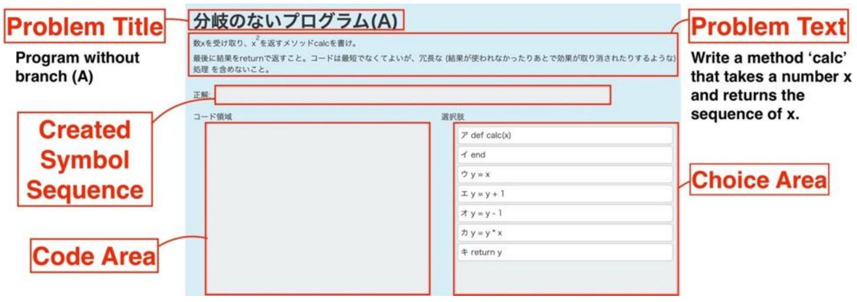
Abstract: This present study investigates the novice programmers using state transition diagrams, to help teachers and teaching assistants (TAs) in their instruction by visualizing the answering process of students. The data for this study are examination results in Ruby programming. Three state transition diagrams, correct answerers, wrong answerers, and to find stumbling points of wrong answerers, are drawn. The subjects of the experiment are TAs in the Ruby programming course. Results show that the subjects identified the conditions of the students by seeing the state transition diagram. This suggests that the visualizing answering processes of students opens a possibility of promoting instruction for teachers and TAs.

Keywords: Programming education, state transition diagram, answering process, split-paper testing, visualization

1. Introduction

1.1 Split-paper Testing

Split-paper testing is composed of fully automatically graded questions (Nakayama, Kuno, & Kakuda, 2020). It was designed to make programming proficiency tests easy to grade. Figure 1 shows the screenshot presented to the students. A test question is given as a correct program divided into lines and indicated as a set of choices represented by a choice symbol. Furthermore, it is mixed with wrong choices. Students must drag and drop the displayed choices onto the answer form to complete the program for the given problem. A log of the choices is recorded when the student answers the question and can be collected from the web system and analyzed as personal data.


Problem Title
Program without branch (A)
Created Symbol Sequence
Code Area
Problem Text
Write a method 'calc' that takes a number x and returns the sequence of x.
Choice Area
正解:
コード領域
選択肢

分岐のないプログラム(A)

数xを受け取り、x^2を返すメソッドcalcを書け。
最後に結果をreturnで返すこと。コードは最短でなくてよいが、冗長な (結果が使われなかったりあとで効果が取り消されたりするような) 処理 を含めないこと。

```
ア def calc(x)
イ end
ウ y = x
エ y = y + 1
オ y = y - 1
カ y = y * x
キ return y
```

Figure 1. Split-Paper Question that is Presented to the Students.

1.2 Fundamental Issues of Programming Education

Many students are faced with difficulties in learning programming languages. McCracken et al. (2001) found students had not learned to write practical programs at the end of an introductory programming course. Lister et al. (2004) attributed this to a lack of problem-solving skills since most students were weak in the skills needed to solve problems. Gomes and Mendes (2007) suggested the lack of understanding of algorithms was a challenge.

Additionally, it is difficult to design an optimal instruction suitable for all students (Essi, Kirsti, & Hannu-Matti, 2005). However, teachers can guide students in building their knowledge and skills through carefully designed materials and approaches.

2. Related Work

2.1 Parson’s Problems

Parson’s problems, which are similar to split-paper testing, have been widely studied. There are many variations of Parson’s problems (Petri & Ville, 2011).

Generally, problems associated with programming exams are classified into two types. One is code tracing, and the other is code writing. There have been comparative studies on code tracing, code writing, and Parson’s problem. Lister et al. (2010) also found a correlation between scores on code tracing and Parson’s problem. Thus, Parson’s problem is an excellent alternative to the traditional code tracing and code writing problems.

Helminen et al. (2012) developed an automatic feedback system and analyzed the solution of Parson’s problem using a state transition diagram. Their study analyzed a conventional Parson’s problem with an indentation operation added. The feedback system sends error messages to students about fewer lines, wrong orders, or wrong indentation. From analyzing the solutions of the students, it was found that the solution paths varied. Indentations were aligned together after fragments of the code with no indentation were aligned first. The results of the analysis are used to improve the automatic feedback system.

Salil and Amruth (2020) proposed an analysis method that focused on the distance between solution paths of Parson’s problems. The analysis was based on the edit distance, which is a modification of the Levenshtein distance. Results showed that students were inclined to difficulties in the few steps leading to the solution.

2.2 Jigsaw Code

Yamaguchi and Oba (2020) developed the Jigsaw Code, which is similar to split-paper testing. Jigsaw Code is analyzed using co-occurrence matrices and found that when two lines are selected almost simultaneously, some relationships existed among them. Using co-occurrence matrices, the jigsaw understood the strategies of students, such as the tendency to place the for-loop chunks first.

2.3 Relationship to This Study

The current study provides direct support for instructing teachers and TAs. Promoting the instruction of teachers and TAs reduces the number of students with difficulties. We analyzed and visualized the answering processes of students using the state transition diagram. The state transition diagram is shown to teachers and TAs during programming courses. This study is positioned as a basic analysis to determine how to visualize the information as the state transition diagram from the analysis of the answering process.

3. Methods

3.1 Preliminary Analysis to Prepare for the Experiment

The method used to analyze the data from the state transition diagram will be designed and discussed. We consider a set of exam questions answered by students for the analysis. The method and results of the analysis are explained in this subsection.

3.1.1 Target of Analysis

A class of 62 (50 males, 12 females, mean age 19.6 years, variance 1.07) fisrt-year university students, in a science and technology university offering a first-year course in programming is our target for analysis. Ten questions in the actual examinations for the Ruby programming language are analyzed with all questions from split-paper testing.

The history of operations, including adding, deleting and changing orders of choices are recorded as logs when students attempt split-paper testing. The CSV files for each student contain the times and answer sequences of all questions, respectively. The sentence and choice for every question as well as the correct answers can be obtained as an XML file. The CSV and the XML files are proceeded by analyzing scripts. Graphviz is used to draw a state transition diagram.

3.1.2 Methodology of analysis

For each question, different types of state transition diagrams are generated for each purpose of analysis. The present study, draws the following three types of state transition diagrams: state transition diagram of correct answerers, state transition diagram of wrong answerers, and state transition diagram to find stumbling points of wrong answerers. These state transition diagrams are created by integrating the individual answers of all students.

State transition diagrams are composed of nodes and paths with either a PDF or PNG file generated as an output of each state transition diagram. A node represents the history of each choice from the initial state (blank) to the final state (answer to be submitted). The start node and the correct answer nodes are represented in gray color. The line weight of the path is proportional to the number of transitions from one node to the next. The line weight of the pass is expressed by the thickness. The label of the path indicates the number of transitions, which is the number of students who get through the path. The size of every node in the state transition diagram of correct answerers and wrong answerers is fixed.

We introduced a stumbling node in the state transition diagram to find stumbling points of wrong answerers. The stumbling node is intended to make it easier for teachers and TAs to identify when the state transition diagram is seen. A stumbling node is defined as a node with two or more wrong answerers among the nodes with correct answerers. The size of the stumbling node is expanded by changing the node radius. It is proportional to the product of the percentage of correct answerers and the number of wrong answerers in the node. Suppose the size of the stumbling node is proportional to the number of wrong answerers alone, then the stumbling node will be oversized, and the graph will be biased. Additionally, a low percentage of correct answerers can have many wrong answerers, which results in a large number of stumbling nodes, which in turn may cause a bias in the graph. The stumbling node is represented by light gray color. Moreover, the color paths are represented in 10 levels according to the number of correct answerers.

3.1.3 Results and Discussion of analysis

Figure 2 is a state transition diagram for a particular problem. From the state transition diagrams of correct answerers, questions with a high percentage of correct answerers are answered in the same order as the correct answerers. This can be because they imagine the whole program. In addition, several correct answerers chose Def and End at first, indicating that the outline of the program was first assembled, then contents were considered. These results may be a useful point for instruction.

The state transition diagrams of wrong answerers show no trend compared with the state transition diagrams of correct answerers. However, some choices lead to the wrong answers. Some of the wrong answerers excluded important descriptions on the control structure, such as End and Return. In order to eliminate simple but important mistakes such as forgetting to insert End, it is considered a key point of instruction to assemble the outline of the program first.

From the state transition diagram to find stumbling points of wrong answerers, it was found that for questions with a low rate of correct answerers, there was little information available in this state transition diagram. In some cases, the red paths coming from the stumbling nodes were connected to another stumbling node. Several choices are important, unlike the correct answerers, the wrong answerers added unnecessary choices or removed necessary choices. The state transition diagrams to find stumbling points of wrong answerers are more difficult to grasp than those of the correct and wrong answerers, respectively. However, we found that node deviated from paths to the correct answer and what choice was wrong.

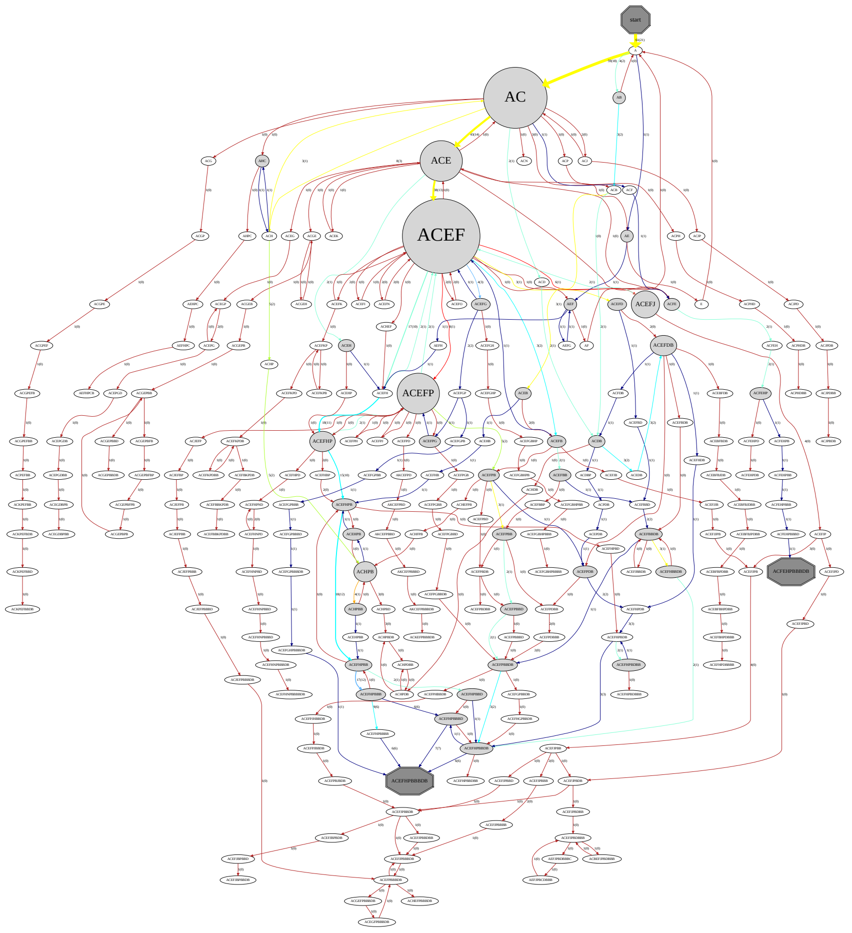


Figure 2. State Transition Diagram to Find Stumbling Points of Wrong Answerers.

3.2 Experiment Design

Figure 3 shows a scene of the experiment. The subjects of the experiment are five graduate students with experience as TAs in the programming course. The subjects examine the state transition diagrams shown on a monitor. It is allowed for the subjects to operate the answer interface using a laptop. They note on the writing paper the findings achieved by examining the state transition diagrams. The questions analyzed are four questions in Ruby language. The percentages of correct answerers for each question differed. This experiment detects the stumbling points of the wrong answerers in the state transition diagrams. It allowed subjects to search for nodes in the state transition diagrams generated as PDF files with inputting answer sequences. The answer interface is the same as the students', allowing the subjects to drag and drop the choices. However, correct answers were displayed at the start. The subjects noted the stumbling points of the wrong answerers, details, and reasons for the wrong answer. After the experiment, questionnaires and interviews were administered.

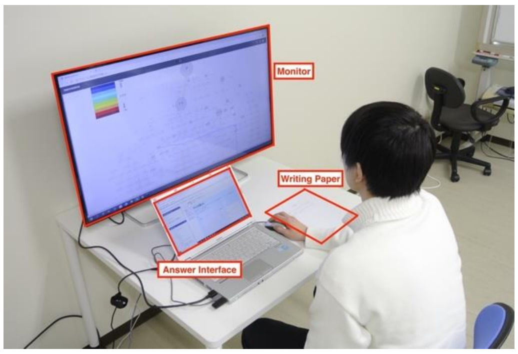


Figure 3. Scene of the Experiment.

4. Results and Discussion

The stumbling points of wrong answerers noted by each subject agree on the same points. The method to visualize the stumbling points is confirmed to be understood correctly.

The list and the results of the questionnaire are shown in Table 1 and Table 2, respectively. All the questions were asked on a Likert scale (1: completely disagree, 2: disagree, 3: undecided, 4: agree, 5: strongly agree). Q1-1 to Q1-4 is asked for each state transition diagram. The average values of the results of Q1-1 to Q1-4 are shown in the Table 2. The Cronbach's alpha coefficient for these questionnaires was 0.86.

Table 1. *The list of the Questionnaire*

Number	Question
Q1-1*	I could read the situation of the answer from the state transition diagram.
Q1-2*	The information shown in the diagram was sufficient.
Q1-3*	I could understand the points where the wrong answerers stumbled from the diagram.
Q1-4*	I was able to understand the contents of the program that caused the students to stumble.
Q2-1	I was interested in the students' solving process.
Q2-2	I saw a trend in their answerers.
Q2-3	I found the diagrams useful in teaching students.
Q2-4	I wanted to use the diagrams in my teaching.

* These questionnaires were asked for each of the four questions.

Table 2. *The Results of the Questionnaire*

Number	TA1	TA2	TA3	TA4	TA5	Ave.	S.D.
Q1-1	4.00	4.00	4.25	3.00	3.50	3.75	0.20
Q1-2	3.50	4.00	4.00	3.50	5.00	4.00	0.30
Q1-3	4.00	3.50	3.75	2.75	3.25	3.45	0.19
Q1-4	4.25	4.75	4.00	2.50	2.50	3.60	0.87
Q2-1	4.00	5.00	5.00	5.00	4.00	4.60	0.49
Q2-2	4.00	2.00	4.00	4.00	3.00	3.40	0.80
Q2-3	4.00	4.00	4.00	3.00	2.00	3.40	0.80
Q2-4	4.00	4.00	5.00	3.00	1.00	3.40	1.36

Table 2 affirms the information displayed in the diagrams as sufficient throughout the experiment. Since subjects showed interest in the answering process of students through this experiment, then visualizing the answering process can be useful for TAs. Although the tendency of the answering process is difficult to grasp, using the current state transition diagram to support the students is TA dependent.

From the interview, many subjects found the stumbling points of wrong answerers by picking up larger stumbling nodes. A few subjects said that it was difficult to learn from the current state transition diagram. Others said that they figured out the correct answerers by looking at the blue bold line in the diagram. As for the answering process of the wrong answerers, it was difficult for the subjects to grasp because of many branches. Generally, many subjects affirm that the current state transition diagram was appropriate for visualizing the answering process. However, a few subjects did not agree. The reason for their position is considered as the amount of information in the state transition diagram being too large and difficult to judge. For improvement, there have been suggestions to reduce the amount of information in the state transition diagram. The current state transition diagram, displays the number of students transitioning between nodes even if it is one student. It is considered as a way of introducing a function such as clustering or turning off nodes with many branches. Another suggestion for improvement is to display the total number of students in the stumbling nodes. Therefore, the current state transition diagram has room for further improvement to make it easier for teachers and TAs to employ state transition diagrams for instruction.

5. Conclusion

The present study visualizes the answering process of students in split-paper testing for novice programmers using a state transition diagram. Analysis and evaluation experiments were conducted. From the analysis, students who answered correctly tended to make up methods, such as Def, End, etc. If the number of correct answerers to a question was high, the answer was assembled in the correct order. For wrong answerers, such students excluded important descriptions for the control structure, such as End and Return. Thus, it is a key point of instruction to assemble the outline of the program first. From the results of evaluating the experiment, it was found that the current state transition diagram has sufficient information on the individual answering process. However, it is difficult to grasp the overall tendency and stumbling points from the information. As pointed out in the interviews, there is room for improvement in the current state transition diagram. This visualization method would be a sophisticated instructing tool for teachers and TAs.

Acknowledgements

This work has been partly supported by the Grants-in-Aid for Scientific Research No. JP19H01710 by MEXT (Ministry of Education, Culture, Sports, Science and Technology) in Japan.

References

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., & Wilus, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. ACM SIGCSE Bulletin, 33(4), 125–180.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. ACM SIGCSE Bulletin, 36(4),119–150, 2004.

Gomes, A. & Mendes, A.J. (2007). Learning to program - difficulties and solutions. 283-287.

Lahtinen, E., Ala-Mutka, K., & Järvinen, HM. (2005). A study of the difficulties of novice programmers. ACM SIGCSE Bulletin. 37. 14-18.

Nakayama, Y., Kuno, Y., & Kakuda, H. (2020). Split-Paper Testing: A Novel Approach to Evaluate Programming Performance. Journal of Information Processing. 28. 733-743.

Ihantola, P. & Karavirta, V. (2011). Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations. Journal of Information Technology Education: Innovations in Practice. 10. 1-14.

Lister, R., Clear, T., Bouvier, D., Carter, P., Eckerdal, A., Jackovà, J., Lopez, M., McCartney, R., Robbins, P., Seppälä, O., & Thomas, E. (2010). Naturally Occurring Data as Research Instrument: Analyzing Examination Responses to Study the Novice Programmer. ACM SIGCSE Bulletin, 41(4),156–173.

Helminen, J., Ihantola, P., Karavirta, V., & Malmi, L. (2012). How Do Students Solve Parsons Programming Problems? - An Analysis of Interaction Traces. ICER'12 - Proceedings of the 9th Annual International Conference on International Computing Education Research. 119-126.

Maharjan, S. & Kumar, A. (2020). Using Edit Distance Trails to Analyze Path Solutions of Parsons Puzzles, Educational Data Mining 2020 (EDM 2020). 638-642.

Yamaguchi, T. & Oba, M. (2020). Measurable Interactive Application to Find Out User Recognition and Strategy when Problem Solving. Journal of Software. 12-22.