Information Engineering Express
International Institute of Applied Informatics
2022, Vol. 8, No. 2, IEE623

Development of a Diagnostic Tool to Verify the Adequacy of Multiple-Choice Fill-in-the-Blank Programming Learning Tasks and the Appropriateness of Their Difficulty Level

Hiroshi Shigematsu [*], Akifumi Ohno [†], Shimpei Matsumoto [*]

Abstract

In programming education at higher education institutions, closed-ended assignments are often presented to learners. Closed-ended tasks are used as a means of checking the learner's understanding level. A typical closed-ended problem in programming is a multiple-choice fill-in-the-blank learning task. When this learning task is applied to a class, the teacher must usually design the problem-solving procedure by themself in advance and verify its validity. Preventing unintended correct answers due to the teacher's oversight is essential, but this task is not easy. Therefore, we developed a diagnostic tool in this study to verify the validity of multiple-choice fill-in-the-blanks programming learning tasks. Using the proposed tool, the teacher can enumerate patterns of input/output according to to answer combinations when creating questions. Since variations in inputs and outputs and overlaps of inputs and outputs are revealed in advance, the teacher can find the difficulty of the learning task and the existence of unintended correct answers before the question is posed. In this paper, we asked teachers to use the proposed tool on a trial basis and obtained feedback from them. As a result, the proposed tool revealed correct answers that the teachers had not noticed before. This defect was derived from a behavior peculiar to the C language and would have been difficult to detect under normal circumstances. Furthermore, the teacher could easily adjust the difficulty level of the problem by knowing the input/output variations results in advance. These results demonstrate the usefulness of the proposed tool.

Keywords: programming education, C language, closed-ended questions, diagnostic tool, fill-in-the-blanks program problem.

1 Introduction

In programming education at higher education institutions and corporate training programs, the fill-in-the-blank programming questions [1][2] (FiB questions) are often presented as a form of learning assignment. FiB questions is one of the closed-ended [4] tasks and is used to check the learner's level of understanding. Closed-ended means that there is always only

[*] Hiroshima Institute of Technology, Hiroshima, Japan
[†] Hello C Development Community, Hiroshima, Japan

one correct answer to a given problem. In a closed-ended problem, the learner does not create all the source code from scratch but rather creates a finished product using pre-prepared materials. Closed-ended problems include source code reading [5], description completion for a part of the entire source code [3], reordering of some descriptions [6], and modification of templates [7]. By nature, programming itself is an open-ended activity. Open-ended means "there is no correct answer" or "a variety of correct answers can exist. However, the open-ended question is difficult to determine the learner's level of understanding of learning tasks because it cannot clearly define the correct answers. Educational institutions need to grasp the learner's level of an understanding case-by-case. For this reason, closed-ended problems are often used in programming education.

FiB questions consist of a test case with input/output examples and a program with blanks in the source code. The learner gives appropriate instructions in the blanks to make the program satisfy the test case. The blanks are based on free-text or multiple-choice. FiB questions are widely used as programming learning tasks and are often employed in certification exams and other paper-based examinations [3][8]. FiB questions allow learners to concentrate on learning and think about the teacher's composition because they do not design the source code themselves. In addition, it is easy to learn, even for programming beginners unfamiliar with computer operations. Therefore, FiB questions is positioned as one of the effective teaching materials in elementary programming education.

While FiB questions are an effective learning tool, they may not promote understanding or make FiB questions too difficult. Adjusting its difficulty level is difficult. The variety of inputs and outputs determines difficulty. The difficulty is also determined by the number of possible response patterns that can yield the correct answer. The inappropriate difficulty level can cause learners to be overloaded or confused, prevent them from concentrating on what the teacher wants them to learn, or give them a sense of overwhelming programming weakness. Therefore, devising a method of setting up the blanks is necessary to successfully promote and deepen the program's understanding when presenting the blanks to the learners.

To deal with the problem of generating appropriate FiB questions, a method to automatically set blanks has been proposed in a previous study [8]-[10], and the proposed method could reduce teachers' burden to create questions. Such a system is useful for teaching the grammar of the program and its basic principles when the learning task does not need to include the knowledge of algorithm learning which is problem-solving procedures in the program's background. However, the teacher has to intervene in setting up the blanks when including an element of semantic content, such as the knowledge of algorithms, is necessary. Specifically, the teacher must select the blanks according to the teaching materials' content by the teacher's intention, so teaching materials are manually prepared in common. When creating such questions, preventing unintentional correct answers due to teachers' overlooking is important in advance. An unintended correct answer may not produce the expected learning effect because the learner may not be active in the teacher's thinking originally envisioned. In particular, when judging comprehension, unintended correct answers may cause learners to acquire the wrong way of thinking. Therefore, avoiding unintended correct answers is necessary as much as possible, but there is limited manual confirmation.

Therefore, this study develops a diagnostic tool to confirm the appropriateness of FiB questions in advance. We asked teachers to use the proposed tool on a trial basis and obtained feedback from them. As a result of practicing the diagnostic tool, we received high evaluations from teachers. The proposed tool revealed correct answers that the teachers had not noticed before. In addition, we could find correct answers that teachers did not intend.

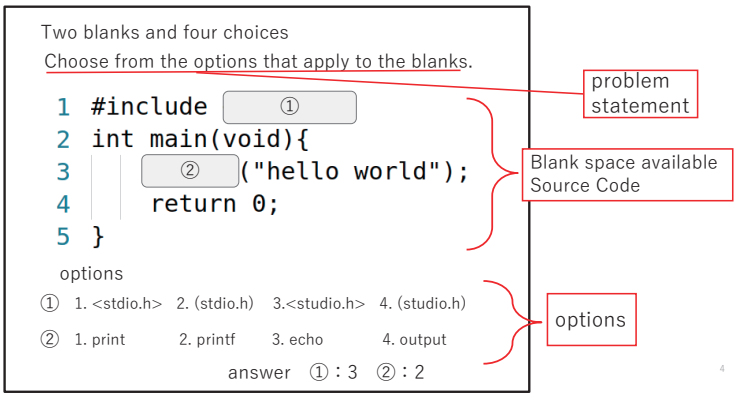

Figure 1: An example of fill-in-the-blank programming question

This defect was derived from a behavior peculiar to the C language and would have been difficult to detect under normal circumstances. Furthermore, the teacher could easily adjust the problem's difficulty level by knowing the input/output variations results in advance. These results demonstrate the usefulness of the proposed tool.

2 Fill-in-the-Blank Programming Question

2.1 Outline

FiB question is considered a useful method to grasp the level of understanding at an early stage [8][10]. As mentioned above, FiB question is a task in which the learner is given a program with input/output examples (test cases) and some blanks in the source code. Then, the learner is asked to think of appropriate instructions to fill in the blanks so that the program behaves as in the test cases.

Figure 1 shows a multiple-choice FiB question. A typical fill-in-the-blank program problem gives learners incomplete source code, as shown in Figure 1, and an input/output example along with the requirement as a problem sentence. Usually, the learner can edit only the blanks and fill the syntax to the presented source code as they deem appropriate. In some cases, the blanks are in free-text format, while in others, they are given in a multiple-choice format. Another method is for incomplete source code to be delivered to the editor as a template, in which case the learner can edit all but the blank spaces.

2.2 Practical examples

In FiB questions, blanks are usually given in the form of choice or the form of an open-ended response. In the case of a program, filling in the blanks by memory is impossible. The assignment requires the learners to collect the necessary information while following the processing flow before and after blanks. Therefore, FiB questions are said to increase the opportunity to think about the program process and promote understanding. FiB questions are also helpful for novices because they are easier to learn than open-ended questions. Indeed, several previous studies have already demonstrated its learning effect, as mentioned below. In some cases, learning tasks are given in an environment where the compiler can

interact with them, and in other cases, they can be completed only on paper or in an LMS. All the presentation methods have confirmed a certain amount of learning effect.

FiB questions are effective in reducing the grading burden on the teacher. FiB questions limit areas to be scored, so automatic grading is easily possible [11]. Other reasons include the immediacy of feedback (the ease of operation when practicing in an LMS such as Moodle) and the ease of learning and evaluation by limiting the structures that teachers can build. If we can automate the grading of FiB questions and the analysis of the results, the teacher can immediately reveal the various levels of learners' understanding. For teachers to provide appropriate instruction to learners in programming, identifying the learners' level of performance is important [12]. For this reason, some exams are conducted to measure learners' programming comprehension. Such exams are used to measure learners' learning outcomes and are administered in various question formats. Among them, FiB questions are one of the most popular question formats. FiB questions are actually used in the prestigious exam for information technology conducted by the Information-technology Promotion Agency, Japan [1].

3 Related Works

Research on FiB questions can be divided into two main categories. One is educational support through system development and practical use [13][14], and the other is research on automatic problem generation [8][15][16].

Kakeshita et al. have developed a support system for FiB learning and have demonstrated its usefulness through practical use [13]. Tana et al. proposed a graph-theoretic blank-element selection algorithm for generating FiB problems and clarified its learning effectiveness through practical use [14].

When creating a FiB question, how to decide the position of blanks is important. Some methods have been proposed to determine the efficient position of blanks. These methods automatically generate FiB questions by deciding which parts of the text should be blanks.

Kashiwabara et al. have proposed a method to identify the key points of the processing flow in a program and designate those parts as blanks using a program-dependent graph (PDG) [9]. PDG is a directed graph showing the dependency between each statement in a program and the data flow. PDG can judge data flow quantitatively, so their proposal method formally sets the blanks without going into the program's semantic content. However, their proposed method is not possible to set the blanks according to the intention of the question's editor. In addition, questions that do not depend on meaning cannot rely on external knowledge. Therefore, the FiB questions generated by their proposed method seem for advanced students who understand grammar. Here, external knowledge means the content the teacher intends for the learners to acquire in the class. For beginners who are the main target of this study, the teacher themselves must design the blanks to reflect their intentions.

Uchida et al. [15] propose a method for automatically extracting keywords and identifiers and filling in the blanks. The target language is Java. First, the teacher registers a question and its description in the system. Then, the system extracts keywords and identifiers from the registered program and generates a FiB question by replacing some parts with blanks. Since the number of blanks is random, different blanks are generated in different

[1] https://www.jitec.ipa.go.jp/index-e.html

places each time, so the same program can generate different questions. This method allows learners to practice repeatedly. But, this method has the disadvantage of not allowing identifiers or reserved words to be specified. Only such capability is difficult to allow for the detailed specification of learning intentions by the questioner's intentions. Besides, this method's blanks do not consider the respective situations of learners and teachers.

Ariyasu et al. ([16]) propose a method for generating FiB exercises based on the teacher's intentions. They generate the questions by analyzing the program and applying the question writer's intentions to it. However, it was unclear whether their proposed method could express the intentions of the problem creators through the experimental result. Also, their study did not mention how to implement the multiple-choice method. As described above, various methods for generating FiB questions have been proposed, but research focused on evaluating the appropriateness of FiB questions created by teachers has not been adequately addressed.

4 The Proposed Tool

Checking the existence of unintended correct answers in a FiB question is not easy in advance. Commonly, the existence of an unintentional correct answer can only be found when a learner's answer, the combination of choices they set, passes the test case. The answer may be correct in extreme cases, even if the choice is completely arbitrary. Though we know that such behavior is always likely to occur, distinguishing which combination of alternatives will contain an unintended correct answer is practically impossible for the teacher. In other words, given a source code with blanks and multiple choices for each blank as a FiB question, the only way to determine if the FiB problem has only one unique correct answer is to fit all the alternative combinations and run them to see. Therefore, we cannot reveal an unintended correct answer without examining all possible response patterns.

Based on the above idea, we newly developed a diagnostic tool to evaluate the FiB question's appropriateness (i.e., to give choices that uniquely determine the answer) by sequentially generating a program to which all possible response patterns are applied and checking the execution results. The number of patterns becomes huge according to the number of blanks in the choices and the problem sentence (the generated source code = the number of choices multiplied by the blanks), so manual verification by teachers is impossible. Therefore, our system first generates source codes with all of the combinatorial patterns. Next, the proposed tool automatically compiles and executes all possible answers to the FiB questions. In this case, all input/output patterns are output to text files. With this diagnostic tool, each choice combination's execution results can be mechanically compared with the teachers' execution examples. As a result, the presence of unintended correct answers can be easily confirmed. The number of source codes generated depends on the number of combinations. If the number is large, unfortunately, it may not be possible to obtain results quickly. in such cases, the teacher must be able to diagnose FiB questions in plenty of time. Despite such limitations, for a typical size FiB problem, the teacher can process it overnight and obtain verification results the following day. Therefore, the proposed tool is useful enough for teachers.

The processing procedure of the proposed tool is shown in Figure.2. First, the teacher prepares the input data (in JSON format) corresponding to the diagnostic tool. The data that needs to be prepared is the problem statement, the choice, the model answer, and the input/output result of the program execution.

The flow of the procedure is as follows.

1. Create the source code using the model choices (input data) and save the execution result (Execution results intended by the question creator) as a character string.
2. Create an overlapping permutation that takes into account the number of choices and the number of blanks. In the case of a problem with three choices and two blanks for each blank, we have the following.we have the following permutation.
{11},{12},{13},{21},{22}{23},{31},{32},{33}
3. Insert the permutation made in 2) into the filled-in-place problem, and generate the filled-in-place problem.
4. Run the generated problem and compare it with the teacher's intended run that created the FiB problem. Only patterns with matching execution results are treated as "Unintended Correctness."
5. Record the choices made for each blank, the generated source code, and the output (or error message if it cannot be executed) on each line.
6. Level the selection pattern for each blank. Details of the level assignment are described in the next section.
7. Display a list of correct answers, including model answers.

First, to find an unintended correct answer in the generated source code, we execute the teacher's model choices and make a model execution result. It is treated as a criterion for comparison with the execution results generated by the brute force method. Next, we create a single matrix of all choice combinations in the input question to generate the source code for all response patterns. For this reason, the number of choices for each blank field has to be the same in the source code verified in this study.

The response patterns for the FiB questions are huge and take a lot of time. Therefore, we thought it was necessary to reduce the processing time. This system uses the standard node.js library "child_process" to generate, compile, and execute the source code in parallel. However, since the number of CPU cores is the limit of simultaneous processes, the number of parallel processes depends on the execution environment.

The execution results are compared with the exemplary execution results as shown in Figure.2 below. Since the execution results are stored as strings, they are added to the correct answers if the results of the comparison match perfectly. Because the brute force method is used to select alternatives, some alternatives contain infinite loops or errors that cannot be compiled. Therefore, in the compilation and execution process, the compilation error source code returns the string "error" as the execution result. The choices are excluded from the correct answer list without exception processing when comparing the execution results. Besides, when the execution result is not returned after a certain period, the proposed tool skips this pattern by processing the execution result in the same way as a compilation error. This study set the time to 100 milliseconds, although it is necessary to change the fixed time in the abovementioned depending on the problem.

Finally, the execution of the generated program's results is compared with the program's execution created using the teacher's model answers. Then, it stores the choices used in the perfectly matched source code as a list of correct answers, outputs them as a result, and tries to find unintended correct answers from the output results.

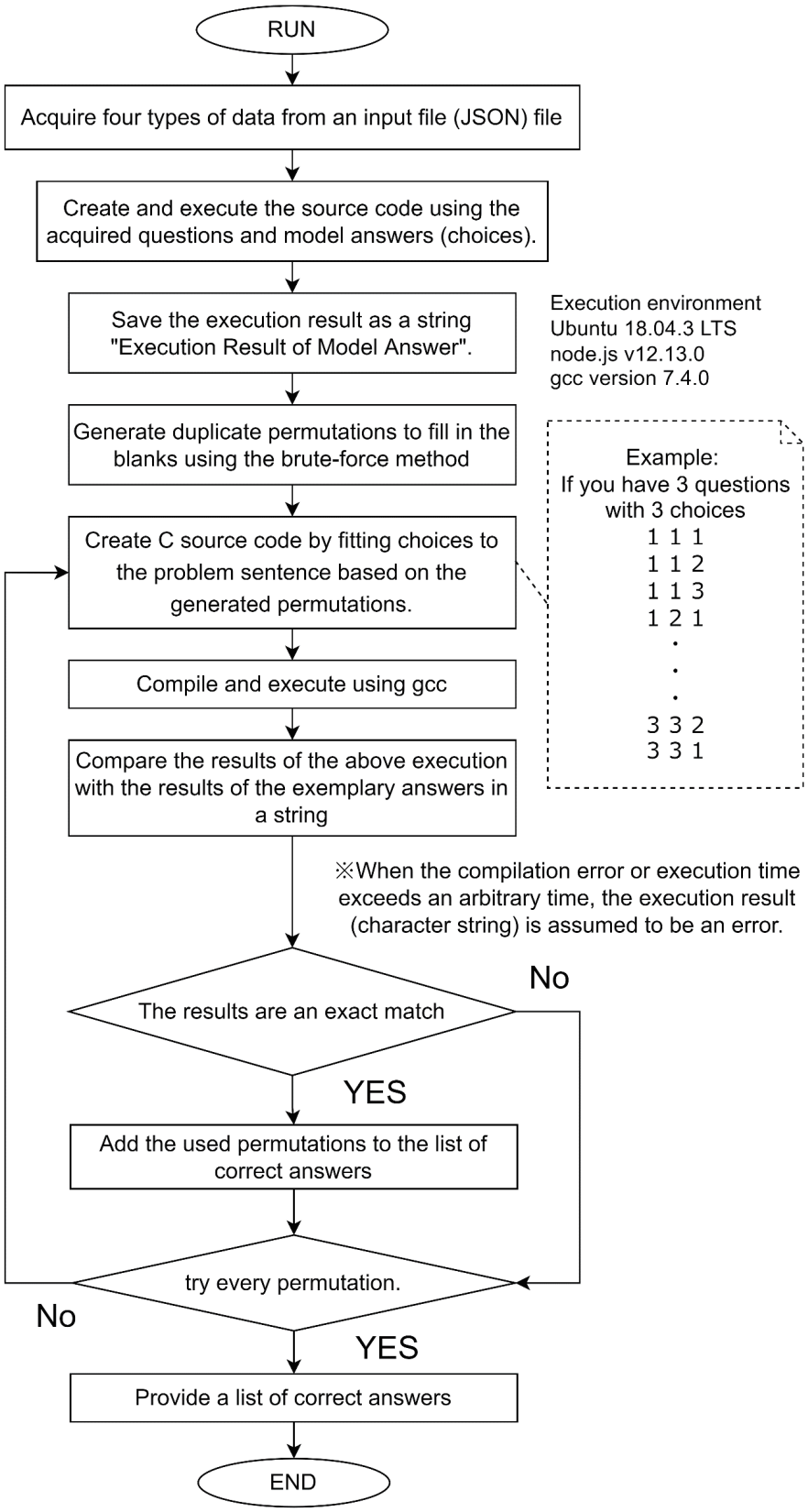

Figure 2: Flowchart

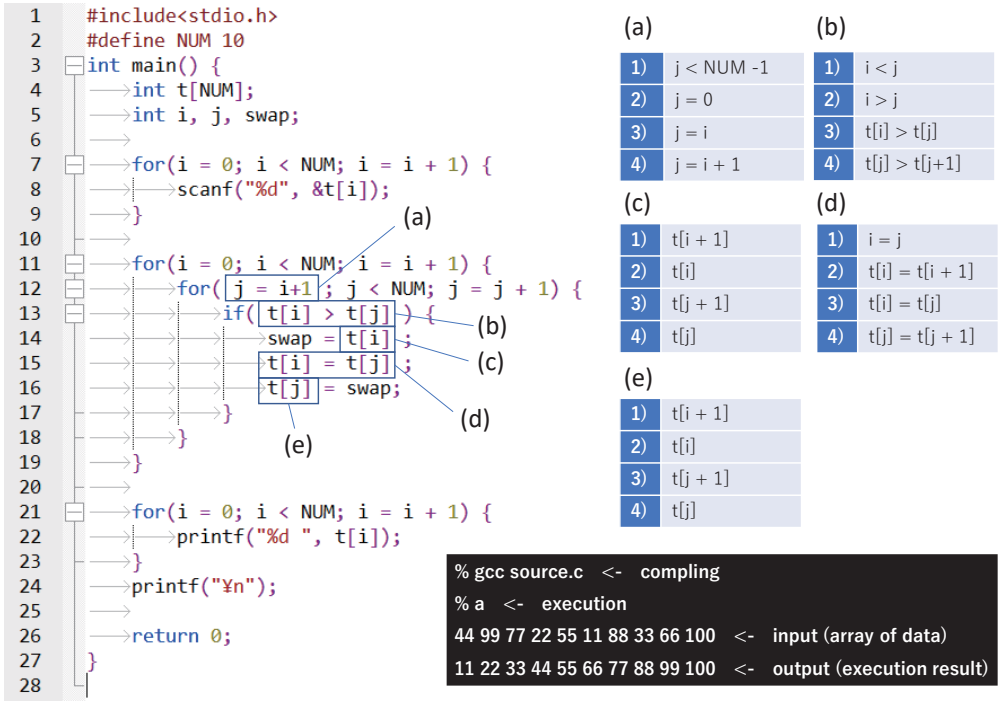

Figure 3: An Example of FiB Question to Show the Usefulness of the Proposed Tool

5 Results

In order to validate and evaluate this diagnostic tool, we used the FiB problem used for programming learning in the lecture to convey the fundamentals of algorithms at a College. Figure 3 shows an example. The question of Figure 3 is a program to sort the inputted values in ascending order. This question has five blanks, each shown in (a)-(e). Figure shows the source code that corresponds to the correct answer choices. This pattern is the choice intended by the teacher. The teacher usually creates dummy choices after defining the correct answer. Finally, the assignment shown in Figure is completed.

The assignment of Figure 3 has five blanks with four choices. Thus, there are 1024 possible patterns, and 1024 source codes are executed by the proposed tool. After the execution, each source code is leveled as described above. The level is determined based on the execution results or on each choice's set. Details of the levels are described below. First, Level 1 is set if either execution does not return results (e.g., infinite loop), a compile error is found, or an out-of-range array index is accessed. Level 1 is the most inappropriate state, which is not satisfactory for execution. If the pattern is not Level 1, it is set to Level 2 or higher. In the case of Level 2 or higher, the level is determined by choice of each blank. The higher the priority blank choice differs from the correct answer, the lower the level. The priority of each blank is given in advance by the teacher. Usually, the blank with the 1st priority is the easiest because most students should answer correctly, and the blank with the last priority is the most difficult. In the case of Figure 3, the priority of blanks is (c), (e), (d), (b), (a). So, if (c) is wrong, the answer is Level 2; otherwise, it is Level 3 or higher. The highest level is 7, the pattern intended as the correct answer. This procedure can be summarized as follows. First, set the initial value of priority as $n = 1$. Then, set the level

$n+1$ for patterns with the incorrect choice in the blank with nth priority. Otherwise, set $n = n+1$ and repeat this process until there are no more unconfirmed blanks.

Table 1: Relative Frequencies of Each Level

Level	Relative frequency
Level 1	0.496
Level 2	0.363
Level 3	0.098
Level 4	0.031
Level 5	0.009
Level 6	0.001
Level 7	0.002

Table 1 shows each level's relative frequencies of all patterns in the learning assignment in Figure 3. As shown in Table 1, the higher the level, the smaller the relative frequency value. However, the value for level 7 is larger than that for level 6. This result suggests that there are two correct answers. That is, Table 1 shows that there exist correct answers that the teacher did not imagine, and that the proposed tool was able to discover them. The combination of choices the proposed tool newly found is (a-3), (b-3), (c-2), (d-3), (e-4), apart from the original correct response pattern (a-4), (b-3), (c-2), (d-3), (e-4). The proposed tool would be useful in that it was able to support such a discovery.

By the way, there was one pattern at level 6. The combination is (a-2), (b-3), (c-2), (d-3), (e-4), and the output results were sorted by value in descending order. In this case, since there is only one way to get level 6, it may be more appropriate to give this combination as the correct answer. We can say that the proposed tool is also useful in that it allows us to present such new perspectives.

The output obtained by the response patterns is used as the explanatory variable to clarify the relationship between the response patterns. Explanatory variables were generated from the strings of the output results. The frequency of each number in the output was tabulated and used as an explanatory variable. With the explanatory variables, the relationship between the patterns was visualized by t-SNE. Patterns that did not return any execution results or compile errors were assumed to be at the origin (vector with all 0 elements). The frequency of invalid values obtained by accessing more than the number of array elements was also added as an explanatory variable. For example, in the case of the learning task in Figure 3, each pattern has 11 elements, and the elements were vectorized by the frequency of each of the ten numbers and one illegal value. t-SNE processed and visualized 1024 vectors. As the hyperparameters of t-SNE, we used 30 for "Perplexity," 1 for "exaggeration," and 15 for "PCA Components," referring to the document [17].

The results are shown in Figure 4. This means that the output of Level 2 and the correct answer are composed of similar strings. Therefore, many students who obtain the output of Level 2 may expect their answers to be close to the correct answers. In reality, however, they are giving answers of Level 2 that are far from correct. If there is a gap between the learner's self-assessment and the actual situation, some feedback should be given to inform the learner of the actual situation. Otherwise, smooth learning may not take place. In this study, the levels of all response patterns were identified. By using the proposed tool, we can resister the information on the levels of all response patterns in the system in advance.

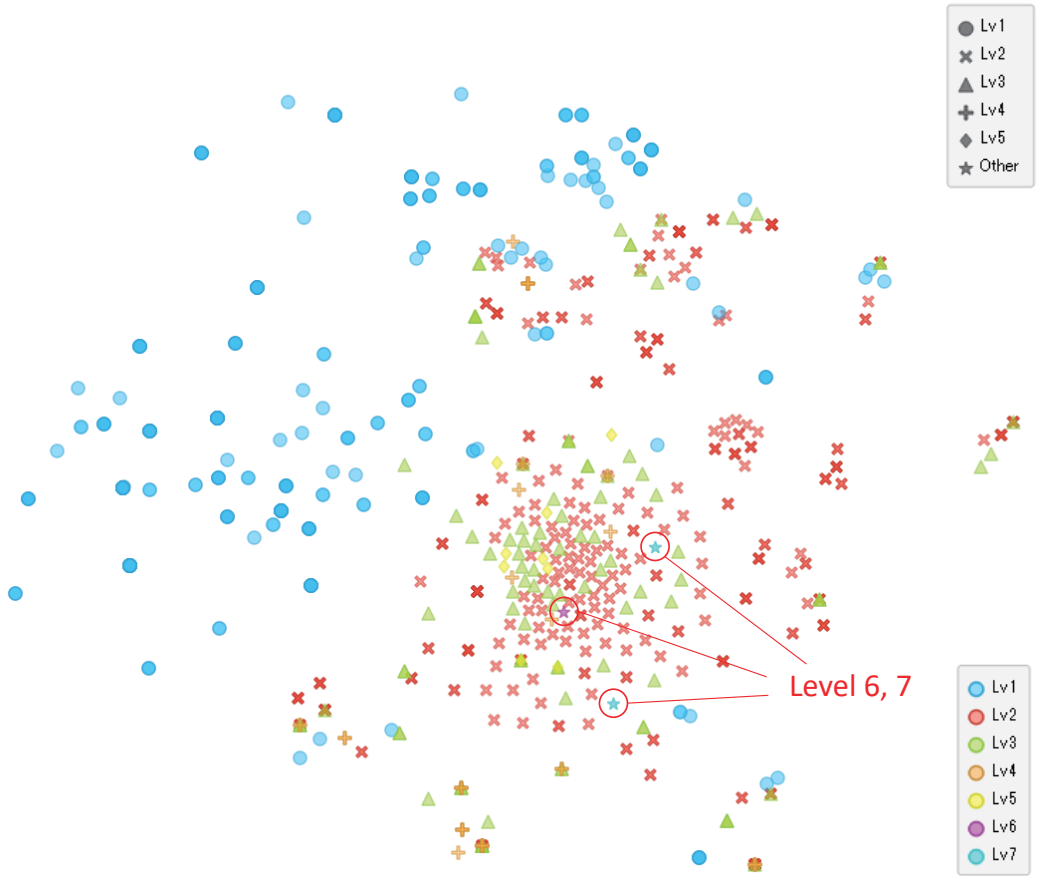

Figure 4: Visualization of Response Patterns by t-SNE

So, when the learner responds to the learning support system of FiB questions, the level of that response can be immediately fed back. This feedback, including the level and its associated information, will be helpful as a scaffold. Scaffolding is useful to support learners' individual learning. For example, the next goal is to show the learner the requirements needed to move up a level (correct answers in the n priority columns) and the output pattern for that level. This information is useful when learning in small steps. Small steps are certainly beneficial for learners who cannot learn by themselves. The diagnostic tool presented new possibilities for this learning in this study. Therefore, this paper has also clarified the pedagogical significance of the proposed tool.

6 Conclusions

In this study, we developed a diagnostic tool to check the appropriateness of FiB questions, including algorithmic learning elements. As a result of testing the usefulness of the proposed tool on a learning task used in an actual class, the proposed tool contributed to the discovery of inadequacies in the learning task. Specifically, the proposed tool pointed out that an response pattern that the teacher had not anticipated could be the correct answer. Furthermore, the proposed tool found a better assignment than the one the teacher had ini-

tially thought of. The teacher was able to actually use the proposal tool and appreciated its usefulness. From the above results, we can firstly confirm that the Proposal Tool is enough useful. Next, this paper confirms the usefulness of the proposed tool for learners. Specifically, this paper showed the possibility of the proposed tool to realize small steps. These results suggest that the proposed tool would be useful not only as a support for tutors, but also as a support mechanism for students. Therefore, the paper concluded that the proposed tool is useful.

Acknowledgment

This work was greatly assisted by Akifumi, Ohno, Hello C development community. Also, this work was partly supported by Foundation for the Fusion of Science and Technology 2018 and Japan Society for the Promotion of Science, KAKENHI Grant-in-Aid for Scientific Research(C), No. 20K03194 and No. 22K02815.

References

[1] Shinkai, J., Hayase, Y., Miyaji, I., A study of generation and utilization of fill-in-the-blank questions for programming education on Moodle. IEICE Tech. Report, et, pp.7-10 (2010), In Japanese.

[2] Funabiki, N., Korenaga, Y., Nakanishi, T., Watanabe, K., An extension of fill-in-the-blank problem function in Java programming learning assistant system. In 2013 IEEE Region 10 Humanitarian Technology Conference, pp.85-90, IEEE (2013), In Japanese.

[3] Kashihara A., Soga M., Toyoda J., A Support for Program Understanding with Fill-in-Blank Problems, Transactions of Japanese Society for Information and Systems in Education 15(3), pp.129-138 (1998), In Japanese.

[4] Pehkonen E., Use of Open-Ended Problems in Mathematics Classroom. Research Report 176, 1997.

[5] Okimoto K., Matsumoto S., Yamagishi S., Kashima T., Developing a source code reading tutorial system and analyzing its learning log data with multiple classification analysis, Journal of Artificial Life and Robotics, Vol.22, Issue 2, pp 227-237 (2017).

[6] Morinaga S., Matsumoto S., Hayashi Y., Hirashima T., A Study on Programming Learning Method to Reduce Non-Essential Cognitive Load, 2018 ISCEAS, International Scientific Conference on Engineering and Applied Sciences, pp.414-423 (2018).

[7] Iwamoto T., Matsumoto S., Hayashi Y., Hirashima T., Examining Presentation Method of Question's Requirement for Game Development-Based Programming Learning Support System, Proc. of AROB 24th, GS5-3, pp.130-133 (2019).

[8] Kashihara A., Kumei K., Umeno K., Toyoda J., How to Make Fill-in-Blank Program Problems and Its Evaluation, Transactions of the Japanese Society for Artificial Intelligence : AI, Vol.16, pp. 384-391 (2001), In Japanese.

[9] Horwitz S., Interprocedural Slicing Using Dependence Graphs, ACM Transactions on Programming Languages and Systems 12(1), 26-60 (1990).

[10] Goshima R., Asai S., Shimakawa Hl, Extraction of Poor Learning Items with Automatic Labeling in Fill-in-the-blank Test. Proceedings of Annual Conference of the Institute of Systems, Control and Information Engineers, Vol. 63, pp. 1503-1510 (2019).

[11] Asai S., Shimakawa H., Automatic scoring system of fill-in-the-blank tests to measure programming skills. In Proc. of the 6th the International Conference on Information Technology and Its Applications, pp.23-29 (2017), In Japanese.

[12] Crow T., Luxton-Reilly A., Wunsche B., Intelligent tutoring systems for programming education: a systematic review. pp.53-62. 10.1145/3160489.3160492 (2018).

[13] Kakeshita T., Yanagita R., Ohta K., Development and Evaluation of Programming Education Support Tool pgtracer Utilizing Fill-in-the-Blank Question, Transactions of Information Processing Society of Japan, Computer and Education, Vol.2, No.2, pp.20-36 (2016), In Japanese

[14] Ta Na., A proposal of blank element selection algorithm for Java programming learning support system, Doctoral dissertation, Okayama University (2017), In Japanese.

[15] Uchida Y., An Automatic Question and Answer System for Learning Elementary Programming, Information Processing Society of Japan, Computer and Education (CE), 2007(123 (2007-CE-092)), pp.109-113 (2007), In Japanese.

[16] Ariyasu K., Ikeda E., Okamoto T., Kunishita T., Yokota, K., Automatic Generation of Fill-in-the-Blank Exercises in Adaptive C Language Learning System, DEIM Forum, D9-5 (2009), In Japanese.

[17] Van der Maaten, L., Hinton, G., Visualizing data using t-SNE, Journal of machine learning research, 9 (11), pp.2579-2605 (2008).

[18] Shute, V. J., Psotka, J. (1994). Intelligent Tutoring Systems: Past, Present, and Future. ARMSTRONG LAB BROOKS AFB TX HUMAN RESOURCES DIRECTORATE.