A Proposal of Recommendation Function for Element Fill-in-Blank Problems in Java Programming Learning Assistant System

Su Sandy Wint, Yan Watequlis Syaifudin, Nobuo Funabiki, Minoru Kuribayash
Department of Electrical and Communication Engineering
Okayama University,
Okayama, Japan
pfpo5v7r@s.okayama-u.ac.jp,
funabiki@okayama-u.ac.jp

Shune Lae Aung
Department of Computer Studies
University of Yangon,
Yangon, Myanmar
shunelaeaung@gmail.com

Wen-Chun Kao
Department of Electrical Engineering
National Taiwan Normal University,
Taipei, Taiwan
jungkao@ntnu.edu.tw

***Abstract*—**

Purpose - To advance *Java programming* educations, we have developed a Web-based *Java programming learning assistant system (JPLAS)*. It offers the *element fill-in-blank problem (EFP)* for novice students to study Java grammar and basic programming skills by filling in the missing elements in a source code. An EFP instance can be generated by selecting an appropriate code, and applying the *blank element selection algorithm*. Since it is expected to cover broad grammar topics, a number of EFP instances have been generated. This paper proposes a *recommendation function* to guide a student solving the proper EFP instances among them.

Design/methodology/approach This function considers the *difficulty level* of the EFP instance and the grammar topics that have been correctly answered by the student, and is implemented at the *offline answering function* of JPLAS using *JavaScript* so that students can use it even without the Internet connections.

Findings - To evaluate the effectiveness of the proposal, 85 EFP instances are prepared to cover various grammar topics, and are assigned to a total of 92 students in two universities in Myanmar and Indonesia to solve them using the recommendation function. Their solution results confirmed the effectiveness of the proposal.

Originality/value The concept of the difficulty level for an EFP instance is newly defined for the proper recommendation and the accuracy in terms of correct answer rates among the students is verified.

***Index Terms*—Java programming, JPLAS, element fill-in blank problem,recommendation function, offline answering function, JavaScript**

Paper type Research paper

I. Introduction

Nowadays, the object-oriented programming language, *Java*, has been widely used in various fields including cloud applications, machine-learning environments, and Internet of Things technologies, due to its high reliability, portability, and scalability. Java was selected the most in-demand programming languages in 2018 [TechRepublic, 2018]. Hence, the strong demand in advancing Java programming educations has emerged from industries. A typical Java programming education in a school, consists of grammar instructions with textbooks in classes and programming exercises using computer operations.

To advance Java programming educations, we have studied a Web-based *Java programming learning assistant system (JPLAS)* [Funabiki, 2018]-[Ishihara, 2017]. JPLAS offers several types of Java programming exercises, starting from exercises for studying code reading and grammar until those for studying practical code writing using object-oriented programming concepts. In any exercise type, each answer from the learner will be marked automatically on the server, to support self-studies of Java programming. Currently, JPLAS provides the *element fill-in-blank problem* [Funabiki, 2017], the *code completion problem* [Kyaw, 2018], the *value trace problem* [Zaw, 2015], the *statement element fill-in-blank problem* [Ishihara, 2015], and the *code writing problem* [Funabiki et al, 2013].

Among the exercise types in JPLAS, the *element fill-in-blank problem (EFP)* is designed for novice students to study Java grammar topics and basic programming skills through code reading. An EFP instance requests a student to fill in the missing elements or blanks in the given source code, which has several blank elements, called the *problem code*. This source code should be of high-quality, most worth for *code reading*. To solve an EFP instance, a student needs to carefully read the problem code and understand the structure, the algorithm/logic, and the semantics.

To support generating new EFP instances by a teacher, the *blank element selection algorithm* has been proposed. It can find feasible blank elements that have the unique answers in the code. Thus, by downloading model source codes from Web sites and applying this algorithm, a large number of EFP instances can be prepared to cover the various Java programming topics for different stages of students in JPLAS. As a consequence, a student needs to select proper EFP instances to be solved that are suitable at his/her learning level.

In this paper, we propose a *recommendation function* of selecting a proper EFP instance to be solved next by the student. To find a proper EFP instance for the student, this function uses the *difficulty level* of each EFP instance and that of the student, and finds the instance whose level is matching to him/her. Besides, it keeps the grammar topics that have been correctly answered by the student before, and selects the instance that has many unsolved topics by the student. The grammar topics are described by the related keywords that are defined in the grammar.

For evaluations, we implemented the recommendation function at the *offline answering function* of JPLAS using *JavaScript*. The offline answering function was developed to support students in solving exercises in JPLAS without the Internet access [Funabiki, 2016]. It runs on a Web browser to offer the same interface as the online version, where the marking and storage functions were implemented with *JavaScript*.

Then, we prepared 85 EFP instances that cover the basic Java grammar topics and have the difficulty levels from 3 to 25, and applied them to the total of 92 undergraduate students in two universities in Myanmar and Indonesia. Their solution results indicate that 1) more than half students reached the highest level while covering around 80% of the keywords, 2) the difficulty level well explains the student performance, and 3) high-level students reached the final one smoothly while low-level students were saturated at middle. Thus, the effectiveness of the proposal was confirmed.

The rest of this paper is organized as follows: Section II reviews our preliminary works to this paper. Section III introduces related works in literature. Sections IV and V present the proposal and its implementations at the offline answering function. Section VI shows the evaluation results of this proposal. Finally, Section VII concludes this paper with future works.

II. Preliminary Works

In this section, we review our preliminary works to this paper.

A. Online JPLAS

In the online JPLAS server, we adopt *Linux* for the operating system, *Tomcat* for the Web application server, *JSP/Java* for application programs, and *MySQL* for the database as shown in Figure 1.

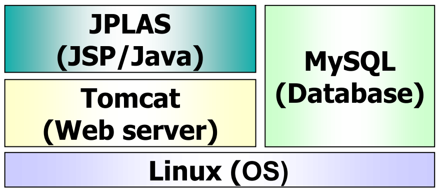


Fig. 1: JPLAS server platform.

JPLAS offers the *teacher support function* and the *student support function*. With the teacher support function, a teacher can create and register new exercise instances in the JPLAS server from the browser, and analyze the solution results of the student including their access records. With the student support function, a student can view the list of the instances assigned by the teacher, select the instances to be answered, and answer the questions in each instance. Each answer from the student will be sent to the *JPLAS* server where the correctness is determined.

B. Offline Answering Function

The offline answering function was implemented for students to solve EFP instances without the Internet access. To avoid cheating by students, the *hash function* for the correct answers and the *message authentication* for the student verification were adopted. The marking function and the data storage function were implemented with *JavaScript* so that they can run at the browser.

C. Element Fill-in-blank Problem

An EFP instance requests a student to fill in the blank elements in the given source code.

1) Definition of Element: An *element* represents the least unit of a code which contains a reserved word, an identifier, a control symbol, and an operator. A *reserved word* signifies a fixed sequence of characters defined in the grammar to represent a specific function. An *identifier* is a sequence of characters defined in the code by the author to represent a variable, a class, or a method. A *control symbol* indicates other grammar elements such as “ . ” (dot), “ : ” (colon), “ ; ” (semicolon), “ (,) ” (bracket), “ {,} ” (curly bracket). An *operator* is used in a conditional expression to describe a condition to determine a logic in a code, such as “<” and “&&”.

2) EFP Instance Generation: A new EFP instance will be generated through the four steps: 1) to obtain a source code that covers the grammar topics to be studied, 2) to divide the source code into a sequence of lexical units or elements and classify the type of each element by applying *JFlex* and *jay*, 3) to select the blank elements that have grammatically correct and unique answers by applying the *blank element selection algorithm*, and 4) to upload the generated EFP instance to the JPLAS server.

3) Blank Element Selection Algorithm: The *blank element selection algorithm* [Funabiki, 2017] selects the elements in the given source code that can be blanked, to generate a feasible EFP instance such that any blank element has the unique correct answer. First, the *constraint graph* is generated from the source code to describe the constraints in the blank element selection. In this graph, each vertex represents a candidate element for being blank and each edge does the pair of two vertices that should not be blanked at the same time. Second, the *compatibility graph* is generated by taking the complement of the constraint graph, to represent the pairs of elements that can be blanked simultaneously. Third, a maximal clique of the compatibility graph is extracted by a simple greedy algorithm to identify the maximal number of blank elements with unique answers from the source code. Finally, extra control symbols are removed from the blank elements so that the ratio between the number of blanked control symbols and that for other elements is controlled.

III. Related Works

In this section, we introduce some related works in literature.

In [Denny, 2009], Denny et al. investigated the quality of the *multiple choice question (MCQ)* repository created by students in an introductory programming course. They analyzed the range of topics on which students chose to write questions without guidance from an instructor. By comparing the repository coverage with a common list of typical introductory programming topics, it was found that students created a repository that covered all the major topics in the curriculum.

In [Hsiao, 2010], Hsiao et al. developed the *JavaGuide* system to guide students to appropriate questions in a Java programming course, and investigated the adaptive navigation support for self-assessment questions in larger classes with a broader range of question difficulty. The topic annotations of the system combine two kinds of adaptation: individual progress-based adaptation and group-wise time-based adaptation, to inform the individual and group-wise importance of the topics. It tries to direct students to the best learning content at any particular moment of time.

In [Brusilovsky, 2013], Brusilovsky et al. presented an attempt to develop a personalized exam preparation tool for Java/OOP classes based on a fine-grained concept model of Java knowledge by exploring two most popular student model-based approaches: open student modeling and problem sequencing. Then, they developed a Java exam preparation tool called *KnowledgeZoom* by combining an open concept-level student model component, *Knowledge Explorer*, and a concept-based sequencing component, *Knowledge Maximizer* into a single interface.

In [Dragon, 2017], Dragon et al. presented the system for automated analysis that can harness the power of online, interactive textbook and practice systems to provide information about high-level conceptual understanding using a *concept graph* to educators. It presents a visualization using *logged data* to provide numeric estimates of a student's knowledge of course concepts, to support teachers and individual students.

In [Hasany, 2017], Hasany presented an e-learning system called *c-Learn*. Students can use it anywhere and at any time as a web application to cope up with the problem that as more lessons are covered, the weak students become weaker in programming. Using this system, a student can gradually learn programming, and in case of errors, it can guide back to the topics which the student should understand to solve the problem.

IV. Proposal of Recommendation Function

In this section, we present the recommendation function of recommending the EFP instance to be solved next by the student. This function can be adopted to other programming languages by changing the keywords and their weights.

A. Difficulty level

To select an EFP instance with the proper level to a student, the *difficulty level* is introduced to each EFP instance and each student.

1) Keyword List: The difficulty level of an EFP instance is derived by the summation of the weights associated with the blanked *keywords* in the problem code. The keywords consist of reserved words, common identifiers, and control symbols related to the Java grammar, referring to [Denny, 2009]-[Hsiao, 2010]. The *keyword list* in Table I lists them. These keywords should be freely used by a student at Java programming study. This list contains the commonly used library classes in Java programming and newly released grammar topics from *Oracle* for the long-term support [Oracle, 2020]-[Java, 2019].

TABLE I: keyword list.

weight	grammar topic	keywords
1	variable	variable
1	access modifier	public, private, protected, default
1	primitive data type	byte, char, short,int long, double, float
1	wrapper class	Boolean, Character, Byte Short, Integer, Long, Double
1	operator	+, -, *, /, ++, - - ==, !=, <, >, <=, >= ?, &, instanceof
1	control statement	if, else if, else, switch, while do while, for, break, continue
1	array	length, declare array
1	common word	class, static, void, main String,System, out, println, in, print
1	code block	{, }, ; ,
2	string	toString,subString,replace, StringBuffer,StringBuilder,split concat,length,indexOf endsWith,startsWith,equals
2	exception	try, catch, finally, throws
2	package	import
2	I/O	Scanner, BufferedReader, File
3	class	field, method, object, constructor this, extends, super, final, return
3	interface	abstract, interface, implements
3	regular expression	Patten, Matcher, compile
3	recursion	recursively called method
3	collections framework	LinkedList, Set HashSet, TreeSet, Map HashMap, Stack, Queue

2) Difficulty Level Calculation of EFP instance: The difficulty level of an EFP instance is actually calculated in the following procedure:

1. Extract any keyword in Table I from the blank elements in the problem code by applying the single keyword pattern matching algorithm [Singh, 2017].
2. Calculate the summation of the weights that are associated with these keywords for the difficulty level of this instance.

In the calculation, the same keyword is counted only once even if it appears in the code multiple times. The following example problem code has seven keywords at the blank elements from _1 _to _7_. They are *class*, *void*, *String*, *for*, < (relational operator), *;* , and *System*. No same keyword is appeared in the example code. Because each keyword has the weight 1, the difficulty level is 7.

Example problem code.

```
public _1_ OperatorSample {
public static _2_ main(_3_ [] args) {
_4_ (int i = 0; i _5_ 3; i++) {
System.out.println(i) _6_
}
System.out.println("i=" + i);
_7_ .out.println("end");
}
}
```

3) Difficulty Level Update of Student: The difficulty level is also calculated for each student. To avoid confusions, the difficulty level for a student is called the *student level* L.

The student level L is dynamically updated by $L + \Delta L$, every time the student solves an EFP instance or gives up it. The level change ΔL is different depending on the *correct answer rate* $x(\%)$ in the previous EFP instance solution. When all the blanks are correctly solved ($x = 100$), ΔL becomes positive. Here, ΔL is changed by the *number of submission times* y, since many answer submissions imply that this student may not understand the previous instance thoroughly. When " Give Up Button " in the user interface is clicked and x is smaller than the given threshold $M(\%)$ ($x < M$), ΔL becomes negative. Otherwise, ΔL is zero.

The procedure of calculating ΔL is given as follows:

- If $x = 100$ and $y \leq N$, then $\Delta L = D$.
- If $x = 100$ and $y > N$, then $\Delta L = \frac{D}{y-(N-1)}$.
- If $x < M$, then $\Delta L = D(\frac{x}{M} - 1)$.
- Otherwise, $\Delta L = 0$.

where D represents the maximum difficulty level change, N does the submission times parameter, and M does the maximum correct answer rate (%), which should be properly assigned as the three important parameters in the recommendation function to select the proper next EFP instance.

B. Next EFP Instance Selection

The EFP instance to be solved next by the student is selected by applying the *instance recommendation algorithm*, when the student solved the previous EFP instance. It identifies the proper EFP instance considering the correct answer rate and the number of answer submissions.

1) Keyword Removal: Referring to [Saroji, 2018], in this paper, we remove the keywords from the keyword list, when the student correctly solved them twice in the previous EFP instances. It intends to cover as many keywords in the keyword list as possible by the recommended EFP instances.

2) Next EFP Instance Selection: Then, the next EFP instance is selected by the following procedure:

1. Find all the unsolved EFP instances whose difficulty levels exist between $L - 2$ and $L + 2$ for the candidates.
2. From them, choose the candidates containing the largest number of solved blank elements that were failed at five or more answer submissions at solving the previous instance, when the correct answer rate x is 100%.
3. From them, choose the candidates from them containing the largest number of unsolved blank elements at solving the previous instance, otherwise.
4. From them, select the next EFP instance containing the largest number of unsolved keywords in the keyword list for the blank elements.

V. Implementation at Offline Answering Function

In this section, we present the implementation of the recommendation function at the offline answering function in JPLAS.

A. Give Up Button

A student may need to know the correct answers of the incorrect blanks, if he/she cannot solve them even after many submissions. For this purpose, " Give up " button is prepared to give up solving the current instance in the interface. When it is clicked, the *JavaScript* program decrypts the encrypted correct answers and shows them to the student. The correct answers are encrypted to be hidden from the students, when the EFP instance is newly generated.

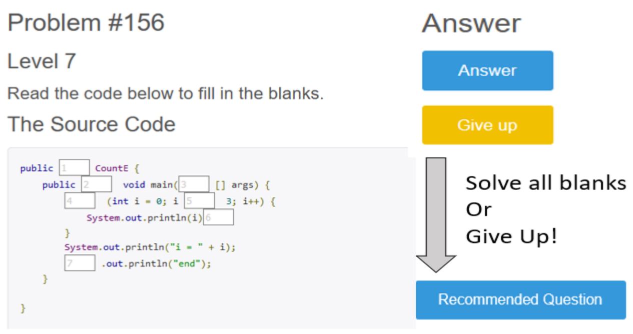
In this implementation, *Advanced Encryption Standard (AES)* algorithm is adopted, where *crypto-js* library [CryptoJS, 2018] is used for JavaScript. The generated secret keys are kept in a file that is not visible from the students by applying the hidden permission access. Besides, we apply *JavaScript Obfuscator Tool* [ObfuscatorTool, 2020] to make the JavaScript codes harder to read, copy, re-use, and modify without authorizations.

B. Recommended Question Button

After the student solves all the blanks or gives up the instance, " Recommended Question " button can be clicked. Then, a new EFP instance will be automatically selected and appeared in the interface. The following procedure is implemented using JavaScript, to recommend a new instance:

1. Initialize the *unassigned instance ID list* by including all the EFP instance IDs, and the *unsolved keyword list* by including all the keywords, when the student starts solving EFP instances. These lists are stored in *localStorage for Web Storage* [WebStorage, 2020].
2. Update the *unassigned instance ID list* by removing the currently assigned EFP instance ID.
3. Update the *unsolved keyword list* by removing the keywords when the corresponding blanks are correctly solved at the second time at the current EFP instance.
4. Update the *student level* by the procedure in Section IV-A3. The *correct answer rates* and the *numbers of submission times* in the previous EFP instance solutions are stored in *localStorage for Web Storage*.
5. Choose all the EFP instances in the *unassigned instance ID list* whose difficulty levels exist between $L - 2$ and $L + 2$ after updating L.
6. Select the EFP instance among the chosen EFP instances that contains the largest number of keywords in the *unsolved keyword list*.

1) EFP Instance Solution Interface: Figure 2 illustrates the user interface at the offline answering function for an EFP instance. It displays the problem number "Problem #156", the difficulty level "Level 7", the problem code with seven blank elements, the answer submission button, the give up button, and the recommendation request button.


Problem #156

Level 7

Read the code below to fill in the blanks.

The Source Code

```
public [1] CountE {
    public [2] void main([3] [] args) {
        [4] (int i = 0; i [5] 3; i++) {
            System.out.println(i)[6]
        }
        System.out.println("i = " + i);
        [7] .out.println("end");
    }

}
```

Fig. 2: Interface for EFP with recommendation function.

A student needs to fill in the proper word in each blank form. If he/she wants to submit the answers for marking, "Answer" button should be clicked. Then, in each form, the background color becomes white if the answer is correct, and pink otherwise. If the student gives up filling in some blanks correctly, "Give up" button can be clicked. After either button is clicked, "Recommended Question" button can be clicked. Then, a next EFP instance will appear in the interface automatically.

VI. Evaluation

In this section, we evaluate the recommendation function for EFP in JPLAS through applications using the offline answering function to undergraduate students in Myanmar and Indonesia.

A. Evaluation Setup

For three important parameters in the recommendation function, we selected $D = 2$ for the maximum difficulty level change, $N = 3$ for the maximum number of submissions, and $M = 80$ for the maximum correct answer rate. Then, we generated 85 EFP instances using source codes in [Denny, 2009] and [Hsiao, 2010] that cover 16 Java grammar topics and contain 110 keywords. The difficulty levels of the EFP instances are distributed between from 3 to 25.

B. Results in Myanmar University

First, we asked 40 second-year students in University of Yangon in Myanmar.

1) Student Solution Result: Table II shows the summary of the student solution results. The 40 students are divided into four groups by the final level and the average number of answer submissions per instance. Besides, the average correct answer rate among the blanks per instance and the average keyword rate among the 110 keywords covered by the solved instances in each group are shown there.

The 29 students (72.5%) in Groups 1 and 2 reached the final level 25 successfully. Thus, they have sufficient Java programming skills. The difference between the two groups is the number of answer submission times. The 16 students at Group 1 answered any instance correctly with up to five times.

The 11 students (27.5%) in Groups 3 and 4, did not reach the final level. They need to improve Java programing skills. Some of them repeated EFP instances for *abstraction* and *regular expression* whose difficulty levels exist between 17 and 20. The three students in Group 4 did not solve many blanks. The correct answer rates are very low, and the numbers of answer submission times are very high. The teacher needs the care of them.

TABLE II: Student solution results.

group	# of students	ave. correct rate (%)	ave. # of submissions	final level	ave. keyword rate (%)
1	16	98-100	1-5	25	79.31
2	13	98-100	5-10	25	77.17
3	8	70-90	5-15	17-20	67.14
4	3	50-60	10-20	13-14	51.67

2) Correlation Analysis Result: Then, we discuss the validity of the defined difficulty level for an EFP instance in this paper. Figure 3 shows the relationship between the difficulty levels and the average numbers of answer submission times by the students to solve all the blanks in each instance correctly for the 85 EFP instances. The correlation coefficient between them is 0.74, which suggests the *strong correlation*.

If an EFP instance is difficult, students will submit answers many times until solving the blanks in the instance. If it is easy, they will solve the blanks with few mistakes. Therefore, the number of answer submission times can be a good index to measure the difficulty of an EFP instance. Thus, the validity of the difficulty level for an EFP instance is confirmed.

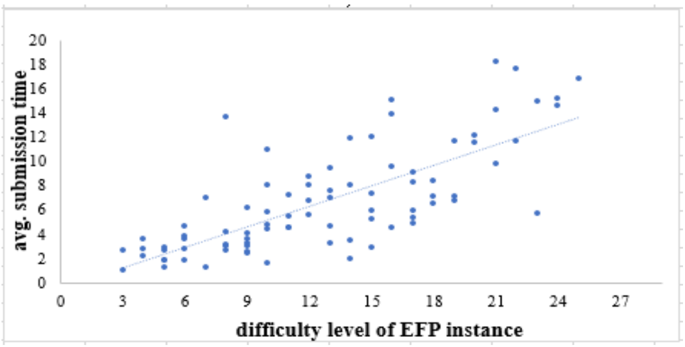


Fig. 3: Relationship between average numbers of submission times and difficulty levels in Myanmar students.

3) Difficulty Level Changes: Here, we observe the difficulty level change of the typical student in each of the four groups during the solutions. Figure 4 shows them in the four groups.

The student in Group 1 continuously increased the level except only one instance until reaching the final level. The student in Group 2 repeated several instances on *regular expression* and *collections frame work* with the similar levels before reaching the final level. The student in Group 3 could not solve instances on *Java string* and *abstract* at levels 17-20 and could not exceed level 20. The student in Group 4 could not solve instances on *I/O* and *inheritance* at levels 13-14 and could not exceed level 13.

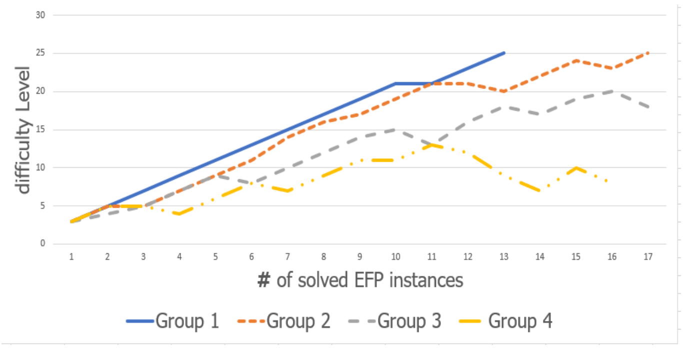


Fig. 4: Typical difficulty level changes of three students.

C. Results in Indonesia University

Next, we asked 52 third-year students in Malang State Polytechnic in Indonesia.

1) Student Solution Result: Table III illustrates the summary of the student solution results. Again, the 52 students are divided into four groups by the final level and the average number of answer submissions per instance.

The 27 students (52%) in Groups 1 and 2 reached the final level 25 successfully. Thus, they have sufficient Java programming skills.

The 25 students (48%) in Groups 3 and 4 did not reach the final level. Some of them repeated EFP instances for *abstraction* and *recursion* whose difficulty levels exist between 16 and 19. The 13 students in Group 4 did not solve many blanks. The teacher needs the care of them.

TABLE III: Student solution results.

group	# of students	ave. correct rate (%)	ave. # of submissions	final level	ave. keyword rate (%)
1	12	98-100	1-5	25	79.67
2	15	98-100	5-10	25	79.07
3	12	06-90	5-15	16-19	60.08
4	13	30-50	10-20	9-15	39.92

2) Correlation Analysis Result: Then, we discuss the validity of the difficulty level for an EFP instance at the results. Figure 5 shows the relationship between the difficulty levels and the average numbers of answer submission times by the students to solve all the blanks in each instance correctly for the 85 EFP instances. The correlation coefficient is 0.77, which suggests the *strong correlation*. Thus, the validity of the difficulty level is again confirmed.

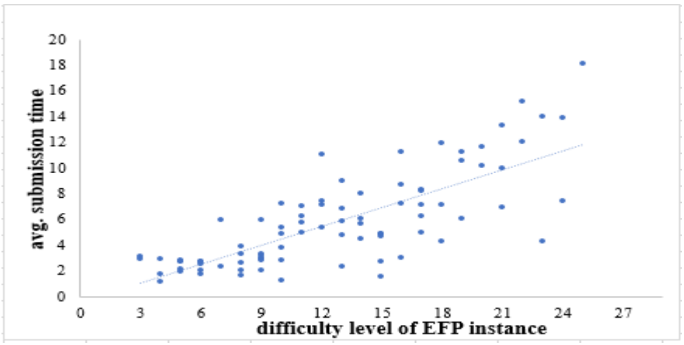


Fig. 5: Relationship between average numbers of submission times and difficulty levels in Indonesia students.

To show the effectiveness of the different weights in Table I, we calculate the correlation coefficient when any keyword is assigned one for the weight. Then, the correlation coefficient for the Myanmar students becomes 0.63 and that for the Indonesia students does 0.61. They are smaller than the correlation coefficients with the different weights in this paper.

3) Difficulty Level Changes: Here, we examine the difficulty level change of the typical student in each of the four groups during the solutions. Figure 6 shows them in the four groups.

The student in Group 1 continuously increased the level without failures. The student in Group 2 repeated a few instances on *Java abstraction*, *regular expression*, and *collections frame work* with the similar levels before reaching the final level. The student in Group 3 could not solve instances on *Java interface* and *recursion* at levels 16-19 and could not exceed level 19. The student in Group 4 could not solve instances on *array*, *String*, and *inheritance* at levels 9-15 and could not exceed level 9.

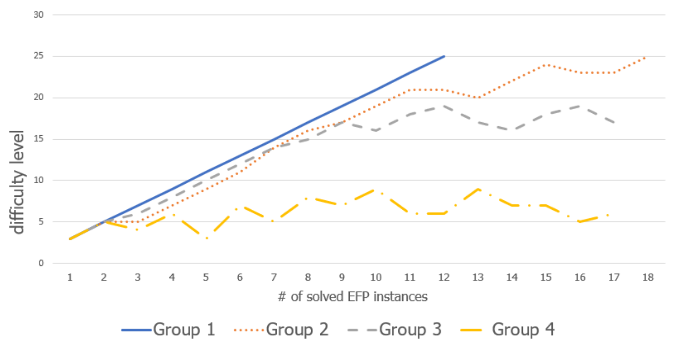


Fig. 6: Typical difficulty level changes of three students.

VII. Conclusion

In this paper, we proposed the recommendation function of selecting a proper EFP instance to be solved next by considering the *difficulty level* of the currently solved instance and the grammar topics asociated with their keywords and implemented it at the offline answering function in JPLAS. Then, we verified the effectiveness through applications to the undergraduate students in Myanmar and Indonesia universities. In the future, we will continue updating the Java grammar topics associated with their keywords, generate more EFP instances, and assign them to students in various universities.

References

TechRepublic (2018, September). *The 10 most in-demand programming languages of 2018*. Retrieved from https://www.techrepublic.com/article/the-10-most-in-demand-programming-languages-of-2018/.

Ao, S. et al. ed.: IAENG transactions on engineering sciences - special issue for the international association of engineers' conferences 2016 (volume II). *World Sci. Pub., 517-530 (2018).*

Ishihara, N., Funabiki, N., Kuribayashi, M., Kao, W.-C.: A software architecture for Java programming learning assistant system. *Int. J. Comp. Soft. Eng. 2(1), (2017).*

Funabiki, N., Tana, Zaw, K. K., Ishihara, N., Kao, W.-C.: A graph-based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system. *IAENG Int. J. Comput. Sci. 44(2), 247-260 (2017).*

Kyaw, H. H. S., Aung S. T., Thant, H. A., Funabiki, N.: A proposal of code completion problem for Java programming learning assistant system. *In: Proc. CISIS. 855-864 (2018).*

Zaw, K. K., Funabiki, N., Kao, W.-C.: A proposal of value trace problem for algorithm code reading in Java programming learning assistant system. *Inf. Eng. Express. 1(3), 9-18 (2015).*

Ishihara, N., Funabiki, N., Kao, W.-C.: A proposal of statement fill-in-blank problem using program dependence graph in Java programming learning assistant system. *Inf. Eng. Express. 1(3), 19-28 (2015).*

Funabiki, N., Matsushima, Y., Nakanishi, T., Watanabe, K., & Amano, N. (2013, February). A Java programming learning assistant system using test-driven development method. *IAENG Int. J. Comput. Sci. 40(1), 38-46 (2013).*

Funabiki, N., Masaoka, H., Ishihara, N., Lai, I.-W., Kao, W.-C.: Offline answering function for fill-in-blank problems in Java programming learning assistant system. *In: Proc. ICCE-TW. 324-325 (2016).*

Denny, P., Luxton-Reilly, A., Hamer, J., & Purchase, H.: Coverage of course topics in a student generated MCQ repository. *Proc. SIGCSE Conf. Innov. Tech. Comput. Sci. Edu. (ITiCSE), 11-15 (2009).*

Hsiao, I.-H., Sosnovsky, S., & Brusilovsky, P.: Guiding students to the right questions: adaptive navigation support in an e-learning system for Java programming. *J. Comput. Assist. Learning 26(4), 270-283 (2010).*

Oracle, *Oracle Java SE support roadmap (2020)*. Retrieved from https://www.oracle.com/technetwork/java/java-se-support-roadmap/.

Biswas, S., Bordoloi M., Shreya, J.: A graph based keyword extraction model using collective node weight. *Int.J. Eng.Tech. 97, 51-59 (2018).*

Oracle, *Java programming language*. Retrieved from https://docs.oracle.com/javase/8/docs/technotes/guides/language/.

MDN web docs, *Using the Web storage API*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Web-Storage-API/Using-the-Web-Storage-API.

JS, *CryptoJS*. Retrieved from https://cryptojs.gitbook.io/docs/.

Obfuscator, *JavaScript obfuscator tool*. Retrieved from https://obfuscator.io/.

Singh, U.: A comparison of single keyword pattern matching algorithms. *Int. J. Eng. Tech. 3(6), 325-329 (2017).*

Brusilovsky, P., Baishya, D., Hosseini, R., Guerra, J., & Liang, M.: KnowledgeZoom for Java: a concept-based exam study tool with a zoomable open student model. *Proc. Int. Conf. Adv. Learn. Tech., 275-279 (2013).*

Dragon, T., & Lindeman, C.: Automated assessment of students ' conceptual understanding: supporting students and teachers using data from an interactive textbook *.Proc. Int. Symp. Multi. (ISM), 567-572 (2017).*

Hasany, N.: E-learning student assistance model for the first computer programming course. *Int. J. Integ. Tech. Edu., 6(1), 1-7 (2017).*

Su Sandy Wint received the B.E. degree in Information Technology from Thanlyin Technological University, Yangon, Myanmar, in 2017, and currently a master student in Graduate School of Natural Science and Technology at Okayama University, Japan. Her research interests include educational technology.

Yan Watequlis Syaifudin received the B.S. degree in Informatics from Bandung Institute of Technology, Indonesia, in 2003, and the M.S. degree in Information Technology from Sepuluh Nopember Institute of Technology, Surabaya, Indonesia, in 2011, respectively. In 2005, he joined State Polytechnic of Malang, Indonesia as a lecturer. He is currently a Ph.D. candidate in Graduate School of Natural Science and Technology at Okayama University, Japan. His research interests include educational technology and database systems. He is a student member of IEICE.

Nobuo Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. In 2001, he moved to the Department of Communication Network Engineering (currently, Department of Electrical and Communication Engineering) at Okayama University as a professor. He was the chairman at IEEE Hiroshima section in 2015 and 2016. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

Minoru Kuribayashi received his B.E., M.E., and D.E. degrees from Kobe University, Kobe, Japan, in 1999, 2001, and 2004. From 2002 to 2007, he was a research associate at the Department of Electrical and Electronic Engineering, Kobe University. In 2007, he was appointed as an assistant professor at the Division of Electrical and Electronic Engineering, Kobe University. Since 2015, he has been an associate professor in the Graduate School of Natural Science and Technology, Okayama University. His research interests include digital watermarking, information security, cryptography, and coding theory. He received the young professionals award from IEEE Kansai Section in 2014. He is a senior member of IEEE and IEICE.

Shune Lae Aung received the B.S. degree from University of Yadanabon, Mandalay, Myanmar, in 2012, and the M.S. degree from University of Yangon, Yangon, Myanmar, in 2015. In 2017, she joined University of Yangon, Yangon, Myanmar, as a lecturer, where currently she is also a Ph.D. candidate. Her research interests include educational technology.

Wen-Chung Kao received the M.S. and Ph.D. degrees in electrical engineering from National Taiwan University, Taiwan, in 1992 and 1996, respectively. From 1996 to 2000, he was a Department Manager at SoC Technology Center, ERSO, ITRI, Taiwan. From 2000 to 2004, he was an Assistant Vice President at NuCam Corporation in Foxlink Group, Taiwan. Since 2004, he has been with National Taiwan Normal University, Taipei, Taiwan, where he is currently a Professor at Department of Electrical Engineering and the Dean of School of Continuing Education. His current research interests include system-on-a-chip (SoC), flexible electrophoretic display, machine vision system, digital camera system, and color imaging science. He is a fellow of IEEE.