Creative Education, 2022, 13, 750-767
https://www.scirp.org/journal/ce
ISSN Online: 2151-4771
ISSN Print: 2151-4755

Process-Oriented Understanding Estimation Using Code Puzzles

Hiroki Ito[1], Hiromitsu Shimakawa[2], Fumiko Harada[3]

[1]Graduate School of Information Science and Engineering, Ritsumeikan University, Shiga, Japan
[2]College of Information Science and Engineering, Ritsumeikan University, Shiga, Japan
[3]Connect Dot Ltd, Kyoto, Japan
Email: hirokiito6900@de.is.ritsumei.ac.jp

How to cite this paper: Ito, H., Shimakawa, H., & Harada, F. (2022). Process-Oriented Understanding Estimation Using Code Puzzles. *Creative Education, 13,* 750-767.
https://doi.org/10.4236/ce.2022.133048

Received: January 20, 2022
Accepted: March 12, 2022
Published: March 15, 2022

Abstract

In the current programming education, in order to assess the true ability of learners, instructors still have no choice but to monitor their answering process, standing by them. However, this is impractical for freshman training in educational institutions and newcomer training in companies. Because of the practicality, a large number of learners are assessed at once using written tests or Web tests. They usually inquire of learners whether they know algorithms and grammar. If not, they assess only the behavior of source codes they submit, at best. Under the training based on such assessment, in reality, not a few learners fail to acquire the skill of writing source codes. It implies that the attainment of programming skills cannot be assessed only by tests on knowledge and submitted source codes. This paper proposes a method for analyzing learners' understanding that focuses on their thinking process of programming. The proposed method focuses on a code puzzle in which learners arrange fragments of a program code to satisfy given requirements. It aims to estimate the learner's perspective on how fragments are built up to achieve the requirements. Learners with low understanding are assumed to be different from those with high in terms of the consistency of arranging ways to compose code fragments for specific blocks in source codes. For the discrimination, the method builds a model using a hidden Markov model. The internal state obtained from this model would help instructors grasp the learner's understanding level. The results of an experiment present that the hidden Markov model produces meaningful values, which enable instructors to interpret the understanding of individual learners.

Keywords

Programming Education, Learning Analytics, Computational Thinking, Code Puzzle, Hidden Markov Model

1. Introduction

The current research trends in the programming education field indicate that many studies are trying to predict performance and learners who drop out (Hellas et al., 2019). However, despite their efforts, there are many learners who drop out of the programming course. Nikula et al. (2011) mention that one of the factors of dropout is motivation, but assessing students' programming abilities. If necessary, we should provide them with appropriate interventions.

Previous researchers have considered that the backgrounds, prior knowledge, cognitive abilities, working time, and learning attitudes may be related to the performance of learners (Luo & Wang, 2020). Especially, many current studies are trying to examine learners' knowledge. However, these information do not allow us to provide appropriate interventions to students. In such programming education for beginners, there are many students who cannot write proper source codes even though they can pass written exams. A true programming ability cannot be assessed only with the learner's knowledge.

Programming skills require not only knowledge but also the ability to construct program elements logically with perspectives. Nesbit et al. who summarized ITS (Intelligent Tutoring System) in computer science and software engineering education also raised alarm over that it has a high percentage of procedural learning objectives and a low percentage of conceptual learning objectives (Nesbit et al., 2015). Under the conceptual learning objectives, learners should attain abilities to build a perspective for the logical construction of program elements, which is referred to programming thinking ability. Since perspectives are built during the assembly of program components, the ability to make perspectives cannot be examined without investigating the process of assembling them so that given requirements should be satisfied.

There is still no established method for assessing programming thinking ability. In the actual situation, the only way to verify the true programming ability of a learner, which consists of both knowledge and programming thinking ability, is for the instructor to stand next to learners to watch their coding. However, in a large class, it takes too much effort to check the understanding of all students in this way. Most of the current educational institutions use measurement methods inquiring learners' knowledge because they are easy to use. This leads to many learners failing to understand the intention of assigned tasks. It prevents them from acquiring the ability to realize them. There is a strong demand for a method to grasp the students' understanding chronologically, taking their programming thinking ability into account.

This study analyses a time-series of the operation process of a learner struggling with a code puzzle in which learners arrange fragments of a program code to satisfy given requirements. The proposed method is to estimate their perspective based on the information of how the code fragments are arranged. The proposed method assumes that differences in learners' understanding are manifested in the consistency of the way to construct code fragments in specific blocks of source codes. The method uses a hidden Markov model to discriminate the

difference.

To confirm the method provides a proper hidden Markov model, the paper examines whether the model can explain the labels given by the instructor. An experiment reveals the trained model can quantify the learners' understanding. The result indicates the model facilitates instructors to detect learners who need additional supervision.

In Chapter 2, the challenges of current programming education are explained, and then the code puzzles and hidden Markov models applied in this paper are described. In Chapter 3, the method for grasping the perspectives in programming tasks using code puzzles is explained, and the method of evaluating is described. In Chapter 4, the experiments and results of the proposed method are shown. In Chapter 5, a summary and future work is discussed.

2. Programming Tutoring System

2.1. Current Programming Education Support

Many of the actual tools to support programming education analyze the learner's understanding based on the results of the learner's answers, while few of them analyze from the viewpoints of the learner's logical thinking. According to schema theory in cognitive psychology (Schnotz & Kürschner, 2007) when humans solve a problem, they read the problem to create a perspective for solving it. It is also the case for the programming field. The logical thinking ability cannot be assessed without considering the way to solve the problem. The paper analyses the understanding in programming, focusing on the thinking process.

In the research area of Intelligent Tutoring Systems (Crow et al., 2018), adaptive feedbacks have become a hot topic in recent years. They are generally classified into those that provide step-by-step hints and navigation, and those that provide summative feedbacks only for the submitted code. It implies the inherent difficulty of developing an understanding of the structure of the program and its associated issues. Nesbit et al. (Nesbit et al., 2015) who summarized the Intelligent Tutoring System (ITS) in computer science and software engineering education also warned about the low percentage of conceptual learning objectives. Villamor (Villamor, 2020) stated the following: Assessing performance-based conventional assessment (McCracken et al. 2001) and assessing only the final outputs do not take into account the student's intentions (Johnson & Gladwin, 1987) and the process that led to their final submission (Lane & VanLehn, 2005), and programming should be regarded as a means of reflecting how students think, decompose, and solve problems, not just generate codes (Marion et al., 2007). As these works point out, in the evaluation of programming ability, it is essential to establish a learning support method that emphasizes logical aspects rather than knowledge.

2.2. Programming Education through Code Puzzle

The proposed method uses the code puzzle as the interface for learning. The

code puzzle is a programming practice task to place the modules or code fragments into the correct order to achieve the requirements. The code Puzzle is inspired by the Parson's Programming Puzzle proposed by Parsons and Haden (2006). The MIT Scratch app that takes a similar approach is well known today. Parson et al. insist that code puzzles are more effective for beginners because they work better at nurturing logical thinking than full-coding. Unlike fill-in-the-blank puzzles, code puzzles require the learner to make the logic flow. It is conceived code puzzle has a format suitable for developing computational thinking, which has been a vital topic recently. Furthermore, the code puzzle is easy to obtain the data of the learner's behaviors such as the actions of choosing, moving, and placing the blocks. These can be used as features to estimate the perspectives of learners. A code puzzle provides learners with an ideal interface of a system for estimation.

2.3. Hidden Markov Model

The Hidden Markov Model is a series of papers proposed by Baum and Petrie (1966). It is a stochastic model which is a Markov process with unobserved (hidden) states. In a general Markov process, the states are observable. The only parameter is the transition probability of the state. On the other hand, in a hidden Markov model, the state is not observed, only the output is observed. The output is calculated probabilistically from the unobservable state of the model. Therefore, the main parameters are the state, the state transition probability, and the output probability from the state. Eventually, it aims to predict the transition sequence of the state that exists behind the observed series.

This paper calls the output as the observed state and the state as the internal state in order to distinguish between the output and the state.

2.4. Related Works

Kato et al. proposed a method to estimate the characteristics of learners from their compiling behavior and time (Kato et al., 2018). The work does not estimate the understanding of learners. Though working time is known to have a large effect on understanding, the analysis proposed in the work provides little information for feedback. Scaradozzi et al. succeeded in identifying various patterns of learners by k-means clustering of visual programming logs, which is similar to code puzzles (Scaradozzi et al., 2020). This method proposed in the work can consider the characteristics of the learners from the means of the clusters, but it does not take into account the process of how the learners solved the problems. Mysore et al. proposed Porta, a web system that can identify locations where learners are struggling in a material book (Mysore & Guo, 2018). Porta computes the location in which students struggled based on their attention to the material book, but does not calculate personal understanding. Since it fails to judge whether the students succeed in the location, it cannot judge misunderstanding points. Guo et al. proposed Codeopticon, an interface, and implementation that supports one-to-many programming learning on the spot (Guo, 2015).

This is a tool that can monitor the learner's code modifications all the time. In a sense, it takes the answering process into account. However, it does not estimate anything on the intention of learners for the modification. It does not provide instructors with an intuitive grasp of whether the learner is struggling or has a perspective. Asai et al. identified the cognitive load of learners and its factors using a fill-in-the-blank task (Asai, 2019). However, the fill-in-the-blank task prevents learners from building the flow of logic. It cannot be said that it takes computational thinking into account.

Most of these studies do not provide estimates of learners' understanding based on the logic. As a paper that focuses on program structure, Blikstein suggested that the understanding of learners can be measured using the clustering method with programming structure (Blikstein et al., 2014). As a study focusing on behavior, Ihantola et al. estimated the difficulty of a task using a decision tree based on the solution process, such as the solving time and keystrokes in programming (Ihantola et al., 2014). They suggested that the solution process has a significant difference in learners' understanding. However, they do not mention the programming thinking ability nor estimate the factors. Another process-oriented approach (Villamor, 2020) is sometimes referred to as WATWIN score proposed by Watson et al. (Watson et al., 2013; Watson et al., 2014) and EQ score proposed by Jadud (2006), which is improved by Tabanao et al. (Tabanao et al., 2011). Both of these are based on compilation behavior. The approach mainly evaluates whether the type of compilation errors has been solved. These approaches seem to be good ideas in accordance with experiences supporting the method proposed in the paper. However, the experiences conclude that compilation behavior and work time do not always represent the learner's ability. The study in the paper decided that it is necessary to consider indicators to assess the learner's ability other than compilation behavior and time. Ito et al. (Ito et al., 2021) work on estimating understanding from the process of working on code puzzles. The work succeeds to estimate 80% of labels given by the instructors to show learners' understanding. However, to estimate perspective based on composing aspects of programming, it is necessary to analyze a time series of operations by learners working on the task.

3. Grasping Perspective Using Behavior of Code Puzzle

3.1. Method for Understanding Estimation Based on Operating Process

The aim of the study is to estimate how good perspective learners have, scrutinizing the process of solving problems along with submitted codes the conventional methods concentrate. This is due to the fact that computational thinking occurs in the process of composing a program. For example, learners who have developed computational thinking should be able to write codes smoothly with a good logical perspective in order to satisfy all the given requirements. To determine whether learners compose programs under the computational thinking, it is necessary to observe their process of writing codes.

Figure 1 shows a schematic diagram of the method. Suppose learners engage in a code puzzle. In the code puzzle, blocks composing a proper program are presented in a random order. The learners are requested to arrange them in a PAD format so that they should compose the program. The code puzzle in this study is implemented as an original application that runs on the web. This application chronologically records various operations on applications as the behavior logs of the learners. The behavior logs should be different depending on the degree of understanding of individual learners.

The following hypothesis is made in the study: The learners who cannot understand the role of each block would touch the blocks randomly. The learners who understand the role of each block would operate certain blocks consistently to assemble them into a module with a specific function. This means that variations in understandings of individual learners can manifest themselves as differences in the consistency with which specific blocks are manipulated. Based on this hypothesis, this paper suggests an educational support system that will help instructors to grasp learners' understanding based on their operation logs.

This method uses a state-space model to represent the learner's process of working on the assignment, as shown in Figure 1. This is due to the idea that the learners' working processes have internal states behind the observable behavior of the learners. The processes can be represented with transition among the internal states, each of which corresponds to their understanding in a specific time point, which blocks the learners manipulate at each time point corresponds to the observable behavior. This method trains a state-space model coincident with observable behavior logs. For a new given behavior log collected from a learner, it estimates which internal state the learner stays. It provides personalized feedback to the learner, interpreting the internal state.

The internal states are expected to be interpreted as the perspective the learner has. This paper investigates whether the estimated internal states can distinguish perspectives of learners with low comprehension from those with low, as instructors standing by the learners do. The estimated internal state would be valid if the internal state variable is coincident with the judgments of the instructors in terms of perspectives. The valid internal states enable instructors to provide new feedback that would be impossible with the traditional assessment based on knowledge or final source codes of learners. The feedback gives learners personalized supervision, which has been inherently difficult to be determined unless instructors stand beside them to watch over their programming processes, as long as the traditional assessment is used. Furthermore, the state-space model calculates the classification probability indicating whether each learner has a high understanding. The probability is calculated so quickly that the learner who needs instruction should be found immediately. In addition, the importance of variables representing the internal states allows us to grasp the differences in behavior between learners with high and low understanding. In other words, it is possible to grasp the points that prevent poor learners from achieving the task.

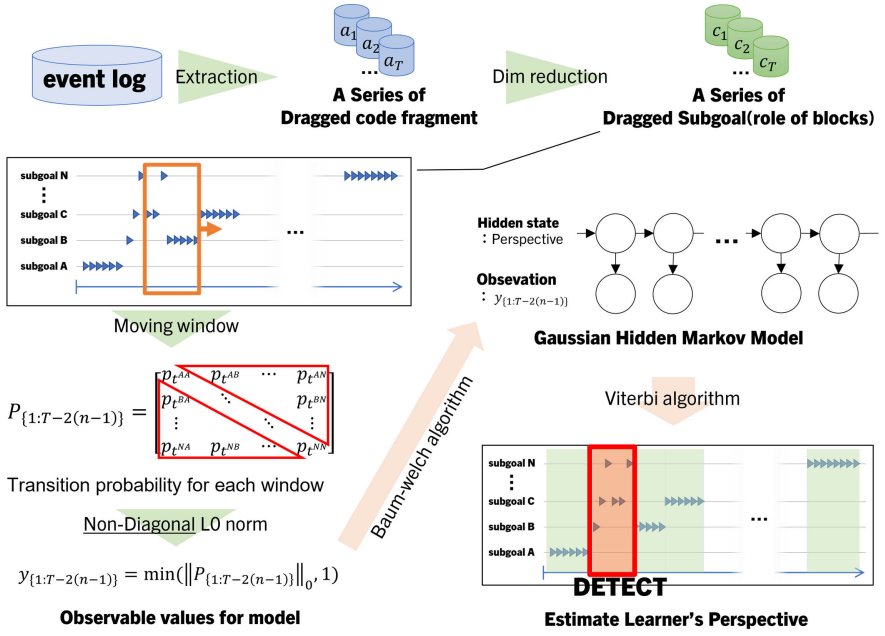

Figure 1. Method to estimate perspective based on code puzzle log.

3.2. Code Puzzle to Collect Learner's Behaviors

This study uses the original web application shown in Figure 2 to collect the characteristic behaviors for estimating learners' understanding. The representation of the program structure follows the PAD proposed by Futamura et al. (Futamura et al., 1981). The web application presents a task in the assignment text screen to make a program whose model source code is given. In the proposed method, the model source code is divided into fragments. Each of them is one or a few lines that play the role of a functional unit. In this paper, the fragments after the division are referred to as blocks. Since operations on every block are collected as behavior, how detailed feedback will be provided depends on the division granularity of the blocks. As shown in Figure 2, blocks are shown to learners to build their programs for each task. Additional tricky or wrong blocks may be added optionally and purposely to test the learners. Learners drag and drop blocks to assemble them so as to satisfy all given requirements in the task.

After learners enter their names, they move to the drawing screen to initiate working on the programming. They can switch between the assignment text screen and the drawing screen using the tabs at the top of the tool. If they have any hesitation during the process, they can check the assignment text again. When they write down their programs to some extent, they can examine the programs on the screen for execution and test. The reason for separating the screens by tabs is to collect data on the time learners spend for each of interpreting the task requirements, creating their programs, and checking and testing their programs. The learners finish engaging the task by pressing the submit button when they conceive to complete the task. The code puzzle system stores whole operations composing of learners' behavior as event data.

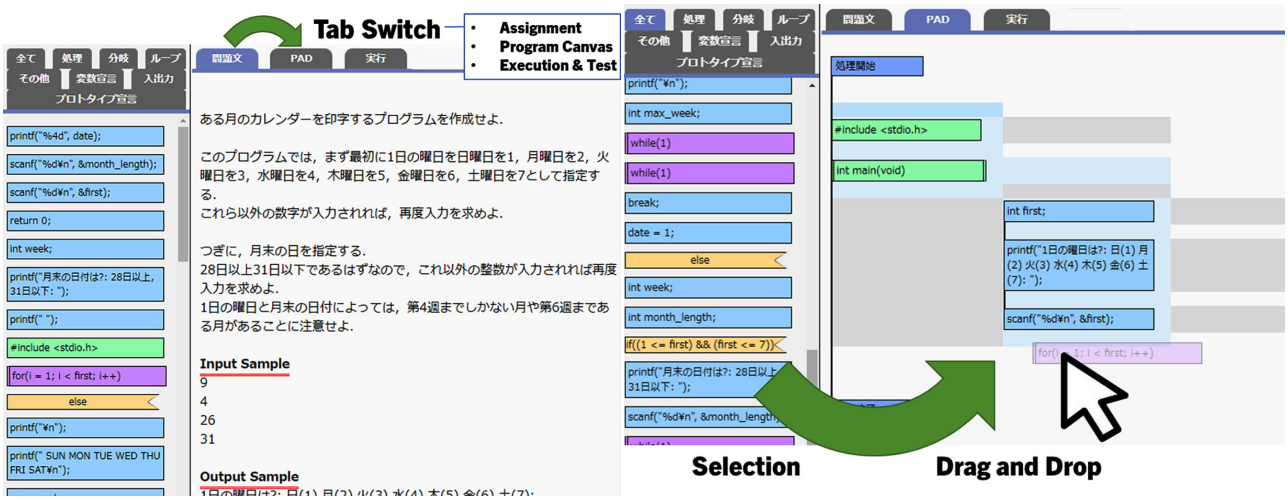

Figure 2. The interface of the code puzzle used in the experiment (The text is in Japanese).

Although this tool collects a wide variety of data, major information discussed in this study is represented with discrete variables indicating which blocks are manipulated. The study builds a state-space model to examine whether each learner has a proper perspective. Even introductory programs used for programming primers generally consist of at least 20 lines. The amount of data collected at composing them is too large for a state-space model to handle. The proposed method pays attention to that each task has several key points learners should achieve. They are referred to as subgoals. In this paper, blocks used in the code puzzles are grouped in terms of subgoals. Blocks are labeled for each group in advance. The labels are then used as observable variables. The roughness with which the consistency of the operation is examined depends on how the subgoals are set. A sub-goal corresponds to a segment containing a series of source code lines implementing a specific logic flow. It is generally set to a program component such as loops, branches, and self-made functions. The proposed method estimates the internal state by training a state-space model from these variables.

3.3. Feature Engineering for Perspective Status

The proposed method estimates the learner's perspective status using a state-space model. The perspective status should not be treated as a single aspect. The task contains multiple subgoals. In addition, the perspectives will change during the process. In order for instructors to grasp the perspective status, it is essential to observe the learner's answering process. In this method, the perspective status is considered in a time series. In other words, the perspective status should be evaluated at each moment when an operation is conducted. The paper considers the manipulation of a certain kind of block as an observable state, and the perspective at the time point as an internal state. Since both the observable state and the internal state are discrete, a hidden Markov model is used as the state space model.

In order to investigate the hypothesis introduced in section 3.1, this section explains how to calculate the consistency in the manipulation of blocks inside a subgoal. In this method, labels assigned to subgoals are used as variables fed to a hidden Markov model. For example, consider when a task has a subgoal of taking input data. The subgoal is implemented with several blocks such as a code to prompt input, functions to get data from the standard input, and a code for error handling. In this paper, these blocks are given the same label, meaning that they are represented with the same value of the variable.

Next, let us consider the transition probabilities from one subgoal to another. When a learner tries to complete achieving a specific subgoal, the learner keeps manipulating blocks in the subgoal for a while. To represent the situation, the transition probabilities are calculated using a moving window. Let us construct a matrix whose rows and columns correspond to the current subgoals and the succeeding ones, respectively. Entries of the matrix can represent the transition probabilities in the time window. When the learner consistently manipulates blocks in a certain subgoal, the diagonal component will be larger in the matrix. Note that a learner who has a good perspective on a specific subgoal manipulates blocks in the subgoal, making few accesses to blocks in others. In this way, the transition probability lets us know the learner's perspective. In order to make a feature that represents whether learners operate consistently blocks in a subgoal, the norm of the non-diagonal entries is calculated. In this case, the norm is not the L2 norm, but the L0 norm defined by the following:

$$\|x\|_0 = \sum_{j=1}^{d} \left| x_j^0 \right|$$

where $0^0 = 0$.

Since the L0 norm indicates how many elements are not equal to 0, it represents the number of other subgoals touched during the moving window.

The larger the norm, the higher the percentage the learner manipulates blocks of different subgoals, which means the learner has fallen into a confused mode, losing a perspective. However, subgoal transition that occurs only once in a moving window is not always a confusion. It may be considered as the moment when one subgoal has been done and the next subgoal has been started. Therefore, a transition of only one time is neglected. The norm of the non-diagonal component of the transition probability matrix is the final observed variable. It is used to train the hidden Markov model. By changing the length of the moving window, it is possible to adjust the severity of the consistency. The longer the moving window, the harder the observed variables get small without a longer consistent manipulation. A long moving window leads to a severe evaluation.

3.4. Explain Understanding Using Perspective

Using the Baum-Welch algorithm, the parameters in the hidden Markov model are trained with the observed variables fed. The internal state at each moment for each individual is estimated by the Viterbi algorithm. Since the manipulation

consistency of each subgoal is used as an observable variable, the hidden Markov model is expected to provide several internal states depending on how the variable changes its value. That is, each internal state expresses whether the learner has a perspective on the subgoal. It is hard to suppose the states transit instantaneously. In this study, three internal states are assumed for the hidden Markov model. The first one corresponds to a state where the learner has a clear perspective. The second indicates the learner has no perspective. The last one is assigned to be a transitive state between the first and the second. The internal state estimated by the Viterbi algorithm for each learner should differ depending on the existence of the perspective. When learners who have no perspective operate on a subgoal, they would frequently experience a state lacking a perspective. On the other hand, a learner who has a perspective would encounter the state corresponding to a perspective many times, while they engage in the construction of the certain subgoal. There may be learners who have no perspective at first, but realize the role of the block in the process. For such learners, the state may switch from one without perspectives to the one having a perspective.

This paper examines the statistical significance of the perspective status estimated by the hidden Markov model. If it coincident with the understanding labels given by instructors, it indicates that perspective status is an important predictor of understanding. The feedbacks made from the estimated perspective are also meaningful.

The following is a method to explain understanding based on perspectives. This paper uses the Random Forest Classifier, assigning the number of occurrences of each internal state and the label by the instructor as the explanatory variable and the objective variable, respectively. This study takes as many explanatory variables as the number of subgoals multiplied by three, since we have set up three internal states.

3.5. Feedback from Perspective

This method can find subgoals for which the learner could not get a perspective through the analysis of the learners' answering process. This cannot be achieved with a system that grasps the learners' understanding from the only their submitted codes. Our method allows to intuitively grasp the learners' progresses on the programming from the manipulated blocks and the internal state estimated using the hidden Markov model. This makes it easy to identify the key points for learners who need guidance. The instructor or automated system can provide a hints or navigation to notify the learners of what to think about next. This can be regarded as adaptive feedback (Kaplan, 2021).

4. Experiment

4.1. Method and Objective

The purpose of this experiment is to clarify whether the learner's operation logs can determine whether they have perspective on a certain subgoal. The subjects

were 20 university students of various levels of programming ability. We did not set a time limit in to obtain the process where they are confused. Some subjects finished the given assignment in 20 minutes and others took 90 minutes. 10 subjects produced correct output while 3 subjects submitted incomplete output. 3 subjects produced slightly incorrect output due to error handling.

Since no time limit was settled, even if the learner was able to produce a correct behaving program in the end, it does not necessarily mean the learner has a high level of understanding. Before starting the assignment, the subjects engaged in a tutorial task to familiarize themselves with the code puzzle. All experiments were conducted online. In this experiment, the assignment was designed for beginning students and included many elements of programming. It is to create a calendar when a day of the 1st and the number of days in a month are input.

4.2. Result of Perspective Estimation

Figure 3 shows the manipulated subgoals and the estimated internal states calculated with the proposed method applied to behavior logs attained from the subjects. Each dot represents a subgoal that was manipulated, while the background color represents the internal state at that time. The red line is a simplified representation of the compilation behavior explained in existing works (Watson et al., 2013; Jadud, 2006). Note that it is not used in the analysis.

In the figure, in case a subgoal is manipulated consistently, corresponding to successive dots in identical subgoals, internal state A appears a lot. On the other hand, internal state C appears a lot, when subgoals are manipulated inconsistently, which is indicated by dots scattered on the subgoals. Internal state B seems to appear as a transient state between them. In this method, transitions that occur only once are ignored. Therefore, the internal states B and C will appear when two or more transitions have occurred in the moving window.

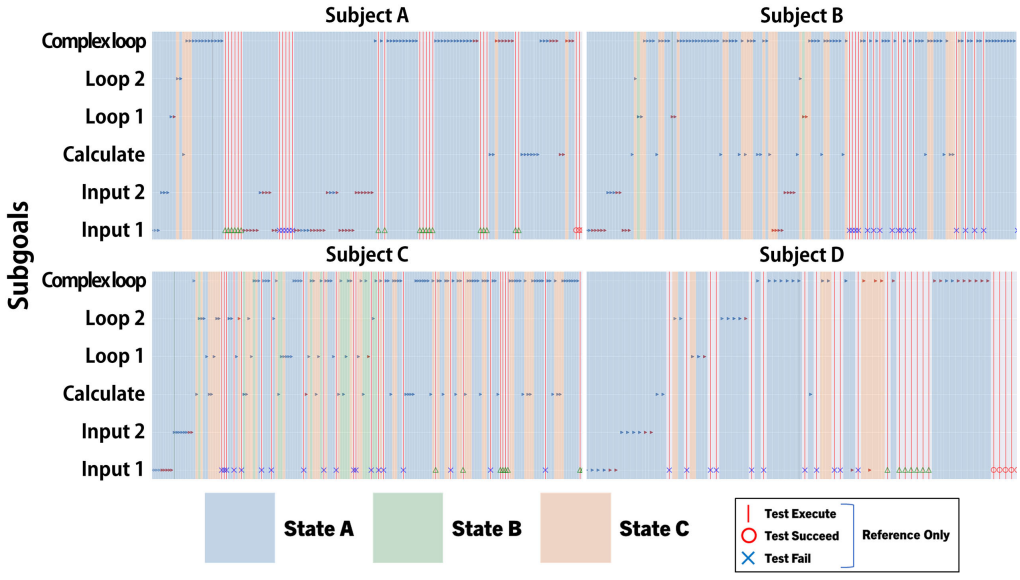

Figure 3. Visualization of perspective and manipulated subgoals.

The information as shown in Figure 3 enables instructors to quickly find out where the learner's confusion occurred. First, subjects with very good programming ability have fewer compilations and a higher percentage of internal state A. In this paper, a particularly interesting result is shown in Figure 3. Subjects A and D were the ones of the learners who performed the solution relatively smoothly. The diagram of subject A consists almost exclusively of internal state A. This means that he understood the meaning of each block enough to compose the program with a clear perspective. Note the existing works taking compiling behavior into account (Watson et al., 2013; Jadud, 2006) regard try-and-error type learners such as the subject D and A as ones with lower understanding. Subject C had difficulty in solving the task. The interpretation of him is simple. After consistently manipulating the "input 1" and "input 2", the other four subgoals were manipulated just like randomly. This is exactly because he did not understand the role of the blocks. Subject D also had difficulty in answering. He seems to have been confused about the "calculate" and "complexloop" subgoals. It should also be noted that even if the learner operates consistently, as in the "complexloop". We can see the perspective may not be correct. For these learners, intervention should be conducted online at the time when internal state C becomes frequent, or when internal state A continues too long compared to the surroundings.

In this way, the perspective state proposed in this paper can be used by instructors or automated systems to notify learners of what they should think about next as a hint or navigation.

4.3. Result of Understanding Estimation

It was found that the internal state estimated by the hidden Markov model has interpretable results. It was also found that it is not possible to say that having a good perspective is a good understanding. Thus, having a perspective on a subgoal is considered to increase understanding in an aspect. On the other hand, if a learner has a wrong perspective, that is, if he or she is confused about how to assemble the certain blocks, the understanding would be rated lower. Therefore, it is expected that the understanding level can be explained by the number of times each perspective status appears in every subgoal.

Let us examine the method presented in Section 3.4. Since one of the 20 subjects misunderstood how to use the tool and did not answer until the end, it is removed as inappropriate data.

Random forest classification was used to generate the model. The explanatory variables are the operations for each subgoal. The objective variable is a binary understanding label attached by three instructors. The labeling was individually conducted in closed sessions with each other. The objective variable was determined by a majority vote. In the evaluation, leave-one-out cross-validation was used. The results are shown in Table 1. The classification performance is 0.894 and 0.875 in the accuracy and the f1-score, respectively. Someone may be skeptical

Table 1. Results of labeled understanding classification.

User id	Classification Probability		Classification Result	
	Low	High	True	Pred
1637246487	0.023	0.977	1	1
1622469590	0.138	0.863	1	1
1622345108	0.151	0.849	1	1
1637056992	0.157	0.843	1	1
1631595816	0.259	0.741	1	1
1637047241	0.269	0.731	1	1
1622434821	0.272	0.728	1	1
1622169741	0.316	0.684	0	1
1631602622	0.512	0.488	0	0
1622145530	0.601	0.399	0	0
1621841538	0.657	0.343	0	0
1 622899809	0.736	0.264	0	0
1631605420	0.745	0.255	0	0
1631600152	0.779	0.222	0	0
1637159166	0.781	0.219	0	0
1631606028	0.785	0.215	0	0
1622430288	0.812	0.188	0	0
1631586473	0.852	0.148	0	0
1622262196	0.912	0.088	0	0

due to the number of subjects. The Repeated 3-Fold Cross-Validation is conducted to get rid of the skepticism. The result was 0.871 for Accuracy.

Figure 4 shows the variable importance. It represents the degree of influence of the explanatory variable on the objective variable. This paper presents the coefficients of the logistic regression in addition to the variable importance of the random forest to investigate the positive or negative degree of influence on the objective variable. The validity of the results is secured because the accuracy showed the same results, although the results of classification are different for a few subjects.

In Figure 4 showing the variable importance, the top four are the subgoals that are relatively easy to solve in the task. All of them are the number of occurrences of internal state A. That is, the more smoothly these subgoals are solved, the higher understanding has been assigned to the learners. On the other hand, internal states B and C appear quite frequently on the other subgoals. This indicates that learners who had no perspective on these subgoals were classified into ones with low understanding. Subgoals Calculate and ComplexLoop have negative coefficients even though they are in internal state A. This means that learners who continue to repeat the manipulations with a wrong perspective are classified into ones with low understanding. However, note that the understanding

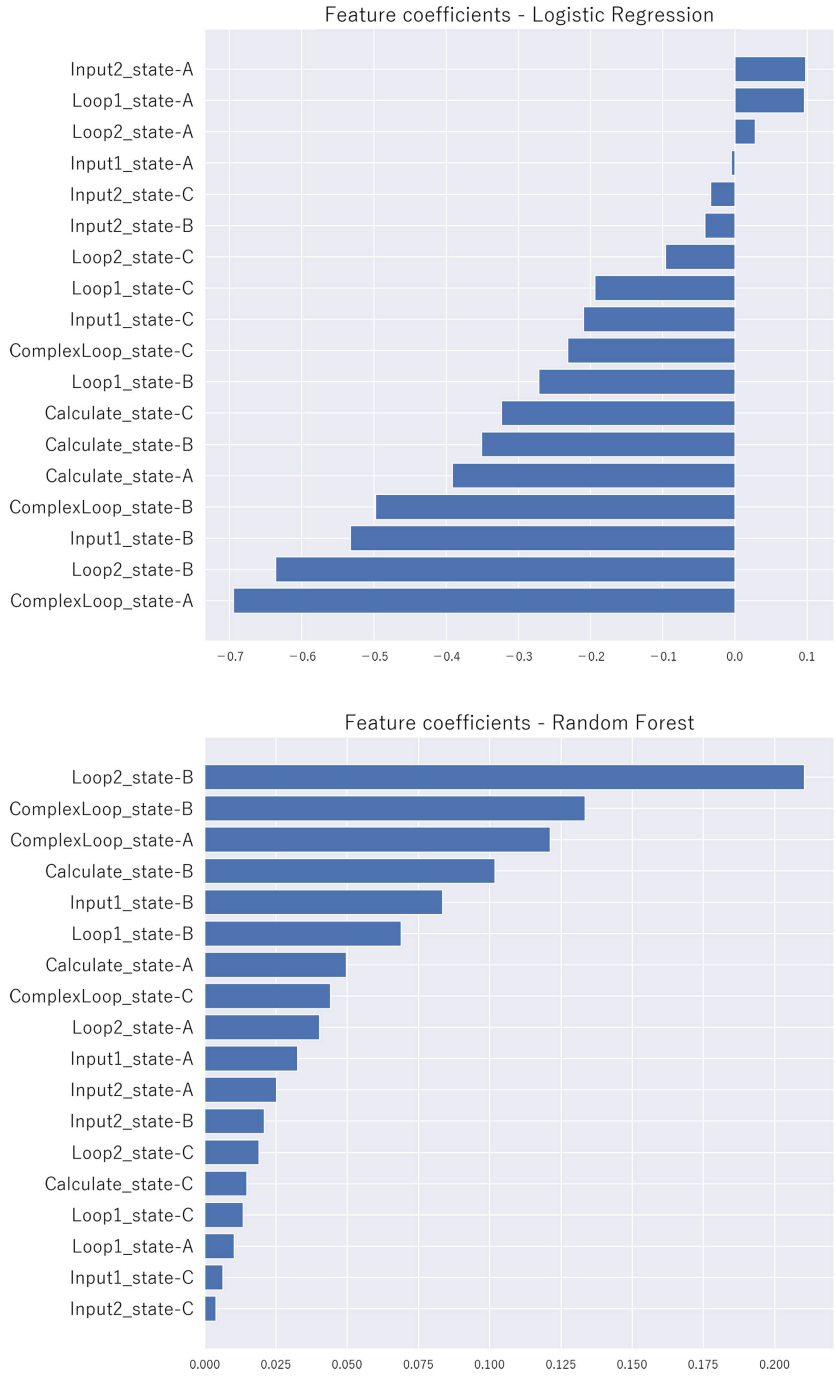

Figure 4. Feature coefficients of understanding estimator.

of learners who manipulate these subgoals without perspective is also estimated to be low (see these internal states B and C).

The results in this section show that the instructors' notion of learners' understanding can be explained by the perspective status proposed in this paper. Furthermore, referring to the results of variable importance, the criteria of understanding discrimination are interpretable. The experiment results indicate the perspective status by the hidden Markov model proposed in this paper is trust-

worthy.

In addition, we conducted another experiment that makes four teaching assistants judge understanding labels referring only to Figure 3. There was a 93% agreement between these labels and the labels used in the classification, though only 15 learners were included. This result implies that instructors can judge the understanding only from the information shown in Figure 3, without standing beside the learners.

5. Conclusion

This paper proposed a method to grasp learners' perspectives from the behavior of answering code puzzles and estimate their understanding from their perspectives. The results of the experiment showed that it is possible to estimate the perspectives on the subgoals with a hidden Markov Model. The subgoal is the key point that the learner should achieve in the assignment. Furthermore, the estimated perspective was able to successfully predict the learners' understanding labels manually judged by more than one instructor.

Such analysis that emphasizes the behavior of the answering process is essential for developing computational thinking, which has been a hot topic recently. Especially nowadays, there are more and more opportunities to give lectures remotely. It is not always possible for instructors to directly watch over their students. In such situations, the system with the proposed method can provide easy feedback to instructors, because that collects and summarizes learners' learning behaviors online. It is also possible to grasp the learner's perspective status on the spot through the time-series analysis. If the proposed method notifies instructors or systems of learners' confusions, they can quickly provide supervision for learners who are suffering from troubles in learning programming, which prevents learners from dropping out. Furthermore, the classification probabilities of the model described in Section 4.3 can be used to arrange the learners' understanding in continuous values. It would enable education providers to use their human resources as much as they can to rescue learners who need help. In addition, our previous method can clarify which code fragments were difficult for the learner, based on the level of attention to each block.

The followings are the future outlooks. Although feedback to the learner in this method is easy due to Figure 3, it has still relied on instructors to check the figure, make a decision, and interpret an estimation model, yet. The goal of our method is to be able to give a kind of summative assessment sheet to learners, almost without relying on any instructor. As an additional try, the existing method of assessing learners' understanding based on compilation errors should be applied to the proposed method, to improve it in terms of providing more feedback. It would be necessary to have a system to exclude those who quit the engagement in the middle of the task in the automated systemization. And, we will practice several assignments and compare the algorithms to see which one is suitable. To collect more various data, this method is to be applied to a variety of assignments.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

Asai, S. (2019). Identification of Factors Affecting Cognitive Load in Programming Learning with Decision Tree. *Journal of Computers, 14,* 624-633. https://doi.org/10.17706/jcp.14.11.624-633

Baum, L. E., & Petrie, T. (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics, 37,* 1554-1563. https://doi.org/10.1214/aoms/1177699147

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming. *Journal of the Learning Sciences, 23,* 561-599. https://doi.org/10.1080/10508406.2014.954750

Crow, T., Luxton-Reilly, A., & Wuensche, B. (2018). Intelligent Tutoring Systems for Programming Education. In R. Mason, & Simon (Eds.), *Proceedings of the 20th Australasian Computing Education Conference* (pp. 53-62). Association for Computing Machinery. https://doi.org/10.1145/3160489.3160492

Futamura , Y., Kawai, T., Horikoshi, H., & Tsutsumi, M. (1981). Development of Computer Programs by Problem Analysis Diagram (PAD). In S. Jeffrey, & L. G. Stucki (Eds.), *Proceedings of the 5th International Conference on Software Engineering* (pp. 325-332). Association for Computing Machinery.

Guo, P. J. (2015). Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. In C. Latulipe, B. Hartmann, & T. Grossman (Eds.), *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (pp. 599-608). Association for Computing Machinery. https://doi.org/10.1145/2807442.2807469

Hellas, A., Ajanovski, V. V., Knutas, A., Ihantola, P., Gutica, M., Leinonen, J., Liao, S. N., Petersen, A., Hynninen, T., & Messom, C. (2019, May 20). Predicting Academic Performance: A Systematic Literature Review. In G. Rossling, & B. Scharlau (Eds.), *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 175-199). Association for Computing Machinery. https://doi.org/10.1145/3293881.3295783 https://research.monash.edu/en/publications/predicting-academic-performance-a-systematic-literature-review

Ihantola, P., Sorva, J., & Vihavainen, A. (2014). Automatically Detectable Indicators of Programming Assignment Difficulty. In B. Rutherfoord, L. Li, S. Van de Ven, & A. Settle (Eds.), Terry Steinbach, *Proceedings of the 15th Annual Conference on Information Technology Education* (pp. 33-38). Association for Computing Machinery. https://doi.org/10.1145/2656450.2656476

Ito, H., Shimakawa, H., & Harada, F. (2021). Advanced Comprehension Analysis Using Code Puzzle. In E. Ziemba, & W. Chmielarz (Eds.), *Information Technology for Management: Towards Business Excellence* (pp. 45-64). Springer. https://doi.org/10.1007/978-3-030-71846-6_3

Jadud, M. C. (2006). Methods and Tools for Exploring Novice Compilation Behaviour. In R. J. Anderson, S. Fincher, & M. Guzdial (Eds.), *Proceedings of the 2006 2nd International Workshop on Computing Education Research* (pp. 73-84). Association for Computing Machinery. https://doi.org/10.1145/1151588.1151600

Johnson, W. L., & Gladwin, L. A. (1987). Intention-Based Diagnosis of Novice Programming Errors. *IEEE Expert, 2,* 94. https://doi.org/10.1109/MEX.1987.4307101

Kaplan, A. (2021). *Higher Education at the Crossroads of Disruption: The University of the 21st Century*. Emerald Publishing. https://doi.org/10.1108/9781800715011

Kato, T., Kambayashi, Y., Terawaki, Y., & Kodama, Y. (2018). Analysis of Students' Behaviors in Programming Exercises Using Deep Learning. In V. Uskov, R. Howlett, & L. Jain (Eds.), *Smart Education and e-Learning 2017* (pp. 38-47). Springer. https://doi.org/10.1007/978-3-319-59451-4_4

Lane, H. C., & VanLehn, K. (2005). Intention-Based Scoring: An Approach to Measuring Success at Solving the Composition Problem. *ACM SIGCSE Bulletin, 37,* 373-377. https://doi.org/10.1145/1047344.1047471

Luo, J., & Wang, T. (2020). Analyzing Students' Behavior in Blended Learning Environment for Programming Education. In *Proceedings of the 2020 2nd World Symposium on Software Engineering* (pp. 179-185). Association for Computing Machinery. https://doi.org/10.1145/3425329.3425346

Marion, B., Impagliazzo, J., St. Clair, C., Soroka, B., & Whitfield, D. (2007). Assessing Computer Science Programs. In I. Russell, S. M. Haller, J. D. Dougherty, & S. H. Rodger (Eds.), *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (pp. 131-132). Association for Computing Machinery. https://doi.org/10.1145/1227310.1227358

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. *ACM SIGCSE Bulletin, 33,* 125-180. https://doi.org/10.1145/572139.572181

Mysore, A., & Guo, P. J. (2018). Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing. In P. Baudisch, A. Schmidt, & A. Wilson (Eds.), *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (pp. 201-212). Association for Computing Machinery. https://doi.org/10.1145/3242587.3242633

Nesbit, J., Liu, L., Liu, Q., & Adesope, O. (2015). Work in Progress: Intelligent Tutoring Systems in Computer Science and Software Engineering Education. In *2015 ASEE Annual Conference and Exposition Proceedings* (pp. 26.1754.1-26.1754.12). American Society for Engineering Education. https://doi.org/10.18260/p.25090

Nikula, U., Gotel, O., & Kasurinen, J. (2011, October 31). A Motivation Guided Holistic Rehabilitation of the First Programming Course. *ACM Transactions on Computing Education, 11,* Article No. 24. https://doi.org/10.1145/2048931.2048935 https://eric.ed.gov/?id=EJ958645

Parsons, D., & Haden, P. (2006). Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In D. Tolhurst, & S. Mann (Eds.), *Proceedings of the 8th Australasian Conference on Computing Education* (Vol. 52, pp. 157-163). Australian Computer Society, Inc.

Scaradozzi, D., Cesaretti, L., Screpanti, L., & Mangina, E. (2020). Identification of the Students Learning Process during Education Robotics Activities. *Frontiers in Robotics and AI, 7,* Article No. 21. https://doi.org/10.3389/frobt.2020.00021

Schnotz, W., & Kürschner, C. (2007). A Reconsideration of Cognitive Load Theory. *Educational Psychology Review, 19,* 469-508. https://doi.org/10.1007/s10648-007-9053-4

Tabanao, E. S., Rodrigo, M. M., & Jadud, M. C. (2011). Predicting At-Risk Novice Java Programmers through the Analysis of Online Protocols. In K. Sanders, M. E. Caspersen, & A. Clear (Eds.), *Proceedings of the Seventh International Workshop on Compu-*

ting Education Research (pp. 85-92). Association for Computing Machinery. https://doi.org/10.1145/2016911.2016930

Villamor, M. M. (2020). A Review on Process-Oriented Approaches for Analyzing Novice Solutions to Programming Problems. *Research and Practice in Technology Enhanced Learning, 15,* Article No. 8. https://doi.org/10.1186/s41039-020-00130-y

Watson, C., Li, F. W. B., & Godwin, J. L. (2013). Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *2013 IEEE 13th International Conference on Advanced Learning Technologies* (pp. 319-323). Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/ICALT.2013.99

Watson, C., Li, F. W. B., & Godwin, J. L. (2014). No Tests Required: Comparing Traditional and Dynamic Predictors of Programming Success. In J. D. Dougherty, K. Nagel, A. Decker, & K. Eiselt (Eds.), *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 469-474). Association for Computing Machinery. https://doi.org/10.1145/2538862.2538930