Algorithmic Thinking Learning Support Syste Based on Student-Problem Score Table Analysis

Mizue Kayama, Makoto Satoh, Kei Kobayashi, Hisayoshi Kunimune, Masami Hashimoto, and Makoto Otani

***Abstract*—The goal of this study is to develop a learning support system for algorithmic thinking for novices. The basic module has been implemented and has been used in computer science courses at some universities. This paper describes some educational functions of this system. The main features of this system are 1) the teacher can control the usage of control elements for developing algorithms based on his educational philosophy, 2) a simple task-answer management system is included and 3) eAssesment functions based on the Student-Problem table method are implemented.**

***Index Terms*—Algorithm describing tool, algorithmic thinking, education for beginners, educational practice.**

I. Introduction

An algorithm is defined as "a set of rules that precisely defines a sequence of operations such that each rule is effective and definite and such that the sequence terminates in a finite time" [1], [2]. As such, until recently, algorithms have been taught in programming courses. However, since about five years ago, algorithms have been taught before programming. There have been many papers on this kind of algorithm and/or algorithmic thinking education, e.g., [3]-[5].

In some research, general programming tools like Java, g++, Scratch [6], BYOB [7] and Squeak & Alice [8] are used as the learning environment. In others, some original learning tools are used to help students express their algorithms. Some tools visualize the student's algorithm, and others animate the student's work.

However, many tools do not have ways to control the learning process, i.e., the teacher cannot customize the tool to suit his/her lesson objectives. Also, many tools do not have ways to evaluate student's work semi-automatically.

In our research, algorithmic thinking is defined as a thinking method to exchange personal ideas about problem solving with others. To foster the formation of mental images of algorithms and data processing, our students should take the algorithmic thinking course before they start to learn programming. We focus on algorithmic thinking as the first step for students in our department. After this experience, they will start to learn about programming and data structures using the C language.

We have proposed an educational method for algorithmic thinking in fundamental education for computer science course students in 2008 [9] and examined the effectiveness of our algorithmic thinking learning support system. We designed our system based on discovery learning [10] and guided discovery [11]. This system uses only three elements as flow structure elements: calculation, selection and repetition. Students describe the algorithm using these three elements.

This paper describes educational functions in our algorithmic thinking system and also discusses some possibilities of our proposed system as an educational tool.

Manuscript received November 6, 2013; revised January 18, 2014. This work is supported by MEXT/JSPS Grant-in-Aid for Scientific Research (KAKENHI): 22300286.

M. Kayama, H. Kunimune, M. Hashimoto, and M. Otani are with Shinshu University, Wakasato 4-17-1, Nagano, Nagano 3808553 Japan (e-mail: {kayama, kunimune, hasimoto, otani}@cs.shinshu-u.ac.jp).

M. Satoh and K. Kobayashi are with Graduate School of Science & Technology, Shinshu University, Wakasato 4-17-1, Nagano, Nagano 3808553 Japan (e-mail: 12tm524h@shinshu-u.ac.jp, kkss@seclab.shinshu-u.ac.jp).

II. Description Styles for Algorithmic Thinking

A. Traditional Styles

What is a suitable description style for algorithmic thinking education for novices? Typical description styles for algorithms or algorithmic thinking are 1) a programming language, 2) a flow chart, 3) pseudo code and 4) natural language. Each style has advantages. For example, a programming language is easily translated into computer readable language, so learners can test whether their descriptions are correct or not. If learners use a flow chart, they can capture the whole logical structure visually. Pseudo code keeps the logical structure of the program, so learners can smoothly shift to some kind of programming language. Natural language is easy for learners to read and write.

On the other hand, they have also disadvantages. As for programming languages, flow charts and pseudo code, learners have to first learn their grammar and notation. Moreover, if learners try to test their algorithms with these styles, there is no allowance for typos, i.e., lack of periods, semicolons and/or parentheses. When learners use natural language to describe an algorithm, there are various interpretations for its meaning.

There are two requirements for the ideal algorithm description style for novices. They are 1) to able to learn the syntax in a remarkably simple way and 2) to foster the understanding of algorithmic thinking. Recently, good solutions have been proposed, i.e., block programming methods such as Scratch. The main features of these methods are:

- No need to learn grammar.
- No notation errors or typos.
- Ability to trace all variables.

DOI: 10.7763/IJCCE.2014.V3.307

- Ability to execute the algorithm stepwise.

However, when we introduced a system using this method to junior high school students and high school students, some pedagogical problems were observed. For learners, there are many choices for appropriate items for the required algorithm, so they seem to be confused as to which item to select. Moreover, there is no support function for algorithm portfolio management of the students. For teachers, they cannot directly add comments or evaluations for students' algorithms. So, the scoring costs for teachers are very high. Therefore, there is uncertainty as to the validity of students' algorithms.

B. Our Proposal

Our research question is "If a teacher can control the usage of elements to develop algorithms, does the cognitive load of students decrease?" To answer this question, we have developed a web based learning system for basic algorithmic thinking education. This system has the following features:

1) The algorithm is represented not by a flow chart, but by columns of blocks such as stacks that grow downward.
2) Input errors can be reduced.
3) The rules to describe the algorithm can be increased in pedagogical stages.

Feature 1 makes learners concentrate on block combinations without thinking about notation. Each block corresponds to all the elements for constructing the algorithm. Learners can insert, move or delete the blocks with their mouse.

To realize feature 2, the variables only need to be typed once. After that, they can be selected from a list. Only operations that are allowed can be selected from a list. Feature 3 allows control of the blocks visible in proportion to the learner's learning progress. This makes learners think about the algorithm in a situation with limited available blocks. On the other hand the blocks that have not been learned are invisible, which helps learners think about the algorithm without stress.

III. Proposed System Description

A. Overview

The system with the features explained in Section II was developed using the Web2pf framework. The combination of some elements represents the structure of the algorithm. Each process is described with a combination of comment statements and a selection of items.

Fig. 1 shows an example screen shot of our algorithm editor for learners. In this system, a user chooses an element for use in the algorithm by double clicking on the editor area. A learner constructs an algorithm with blocks which were chosen. Executable parts are connected to the "start" block (upper left area in Fig. 1). The sets of blocks that are not used can also be put in this editor area. They are shown in gray. A learner can watch the change of the variables values that he/she uses in his/her algorithm. Moreover, some output data can also be watched in this editor.

There are two kinds of blocks, structural blocks and commentary blocks. The structural blocks are calculation, variable declaration, selection, repetition and break. The selection block and repetition block can be nested in other blocks. The commentary blocks are planning and comment. The planning block can be nested in other blocks including structural blocks.

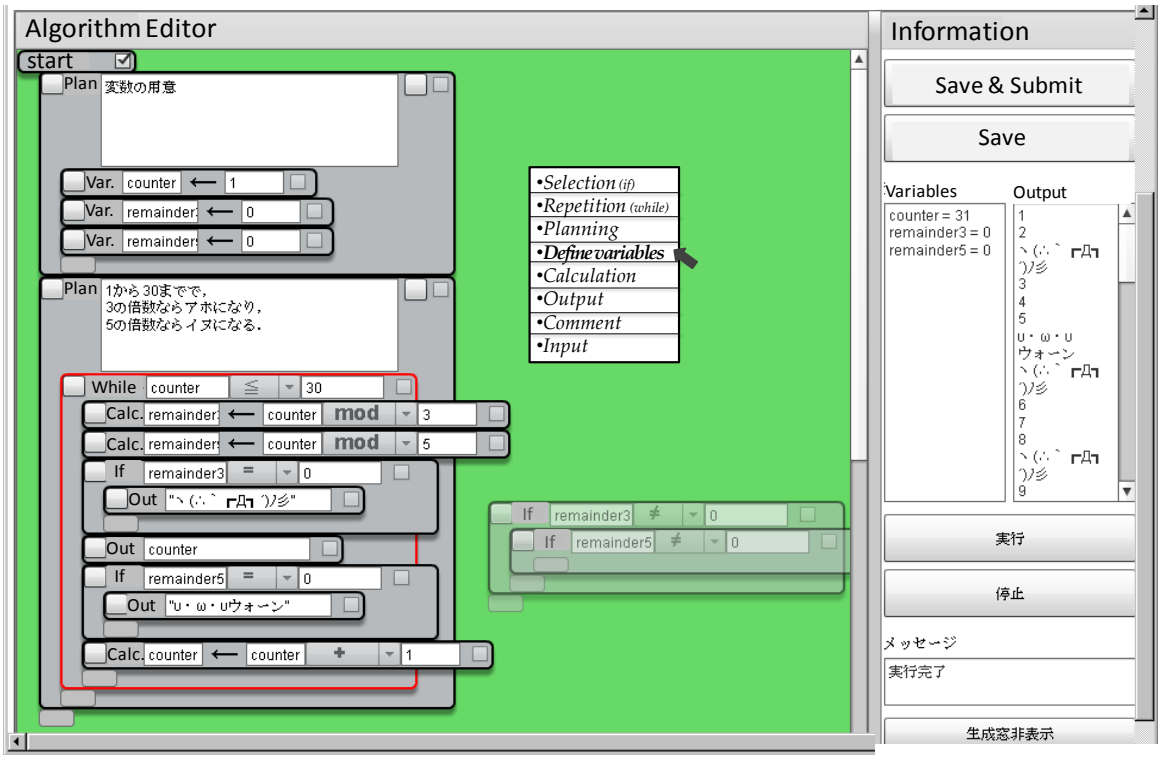


Fig. 1. The algorithm editor of our tool.

Fig. 2 shows the window where rules can be defined for describing the algorithm and the selectable items for the learners. A teacher can define the description rules for each algorithm question. He/she only needs to click the check boxes shown in Fig. 2, and then saves this rule with a name. When he/she submits a question, one description rule can be

attached.

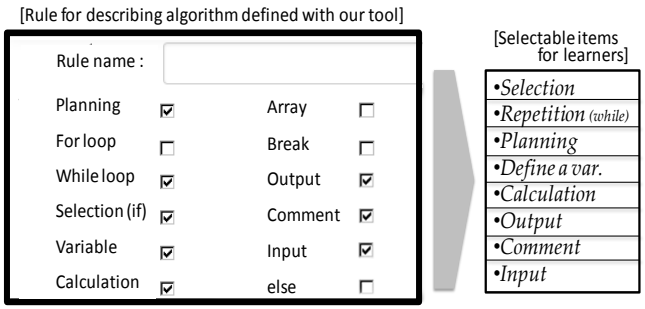


Fig. 2. Definition of the rule for describing the algorithm.

The teacher usually plans the learning steps for his students. As an example, suppose that the first step is a calculation by assignment and/or arithmetic operations with abstract numbers and/or variables, the second step is a conditional test with relational operations and the third step is an iteration with termination conditions. Corresponding to these steps, the algorithm description rules can be defined. Fig. 3 shows this set of learning steps and the description rules.

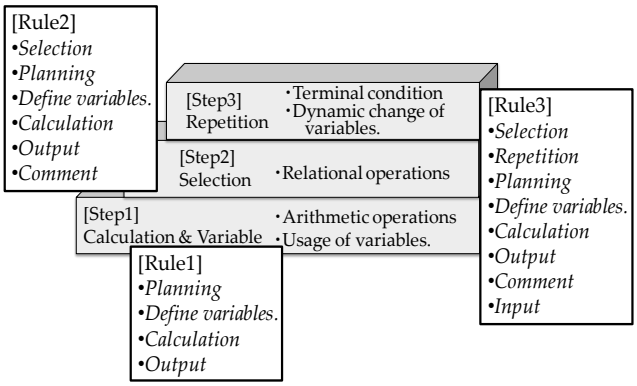


Fig. 3. The learning steps and the algorithm description rules.

B. Pedagogical Features

The main pedagogical features of this system are as follows:

1) Simplify the learning items

Because our subjects are novice learners, only integer variables are used and only one or two-dimensional arrays can be processed. Therefore, the input data type is "variable". However, the output data types are "variable" and "string". Some kinds of messages can be used in our learner's algorithm. The calculation type is limited to binary operations. The operators that can be used are the five arithmetic operations (+, -, *, / and % [remainder]) and the relational operations (<, <=, >, >=, ==, !=). The flow structures are "selection" and "repetition". As for repetition (iteration) operations, two types of elements: iteration with loop time (for type) and iteration with exit/continue condition (while type) are provided. The "break" operation, which can be used to exit from an iteration, is also included (see upper right part in Fig. 1).

2) Teacher can control the usage of control elements

Each teacher has his/ her own policy to teach algorithmic thinking. Each policy has to be appreciated because a teacher has a responsibility for his/her course. Therefore, by using our system, teachers can control which elements are visible according to the class rules. The three steps in Fig. 3 are attached different algorithm building elements. The rule set can be independently managed for each step. A teacher can set this rule before his/her lesson. This rule can be changed during the lesson.

Moreover, a teacher can define the permission of learner's activities for constructing algorithms, i.e., permit to submit/save/execute/edit/syntax check, show program code, log operation history or not. In our system, those things are defined as controllable elements. By using this function, the teacher can define an algorithm reading task (not to permit learners to edit and run), an algorithm reading and running task (permit learners to run) and a scaffolding task (give a template algorithm using planning blocks with teacher's comments).

Teachers can also decide whether the trace function is available and its interval. Teachers can make the trace function available to debug the algorithm or to make it unavailable to make learners think about tasks and learning progress. If a teacher gives permit to use this function, learners can trace the execution of the algorithm and specify a variable when tracing. The right area in Fig. 1 shows a variable trace area and the result of algorithm tracing (output area).

3) Template for each task

This allows learners to see a part of the algorithm or the whole algorithm on their monitor. Therefore, combining this function with the learning activity control function, learners use this system to dissect the algorithm, finish incomplete parts of the algorithm, select the suitable algorithm from some choices, and describe the algorithm from scratch.

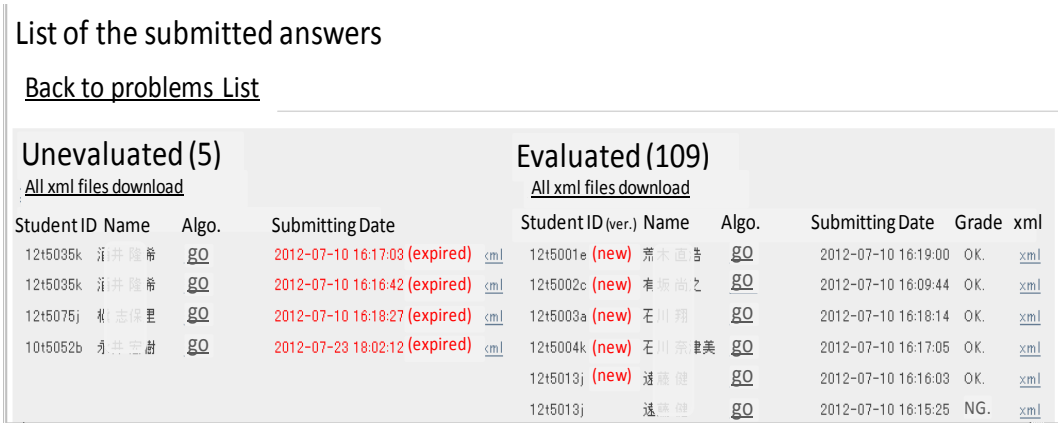


Fig. 4. Statement management of learners' reports.

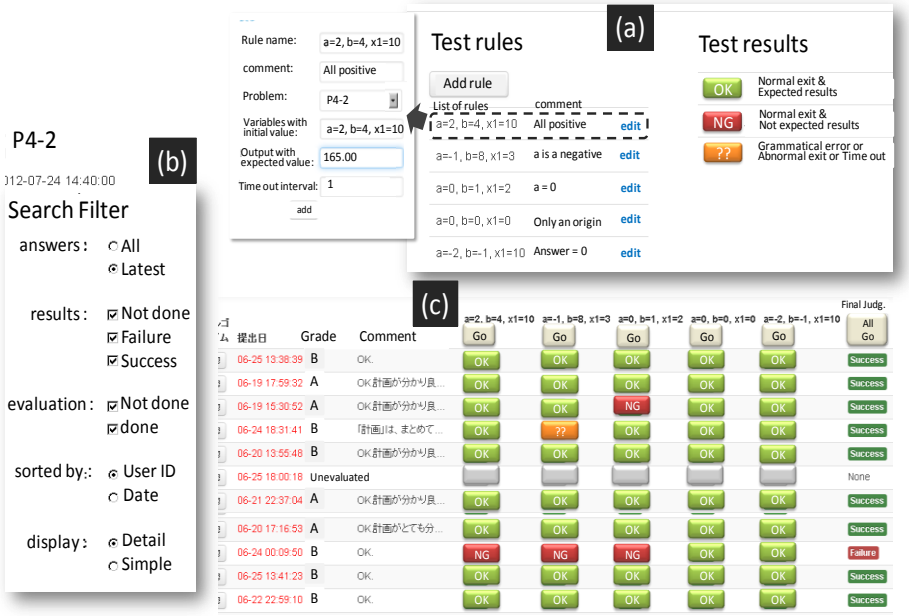


Fig. 5. Auto testing for students' answers.

4) Simple task-answer management system

a) State management of learners' reports

This system helps teachers and learners manage the report status for each learner and each task. For instance, saved but not submitted / submitted, rejected / accepted, unevaluated / evaluated. An example is shown in Fig. 4. In this figure, students' answers are divided into two columns: unevaluated and evaluated. In the evaluated answers, a grade for each answer is shown. This status shows whether the students' answer is rejected or accepted. In this case, two grades are used: OK and NG.

b) Semi auto test for the submitted answers

If a teacher defines the test rules for a problem, our system tests students' answers automatically. An example of this function is shown in Fig. 5. Fig. 5(a) shows the test rules and test results. Currently, the test rule is defined as a set of variables (*a*, *b*, *x1*) with initial values (2, 4, 10) and output variables with expected results (165.00).

There are three types of test results: 1) normal exit and the algorithm output the expected results (OK), 2) normal exit but the algorithm does not output the expected results (NG), 3) grammatical error or abnormal exit or time out (??).

Fig. 5(b) shows the filter to display students test results. In this case, the latest answers and all the results which are evaluated and unevaluated are displayed in detail sorted by user ID.

Fig. 5(c) shows the test results using this filter. The test results for the five test rules defined in Fig. 5(a) and the final judgments in the right column are listed. The teacher can also define how to summarize total test results for the final judgments for the students' answers. In this example, at least three rules have to have an OK status.

5) eAssesment functions based on student-problem score table analysis

The student-problem (SP) score table analysis is an educational analysis method based on students' responses, i.e. test score patterns [12], [13]. The SP score table is a two-dimensional table where rows are student numbers and columns are problem (i.e., test question) numbers. In the table, if a particular student answers a particular problem correctly, the cell is filled with a 1. Otherwise, the cell is filled with a 0.

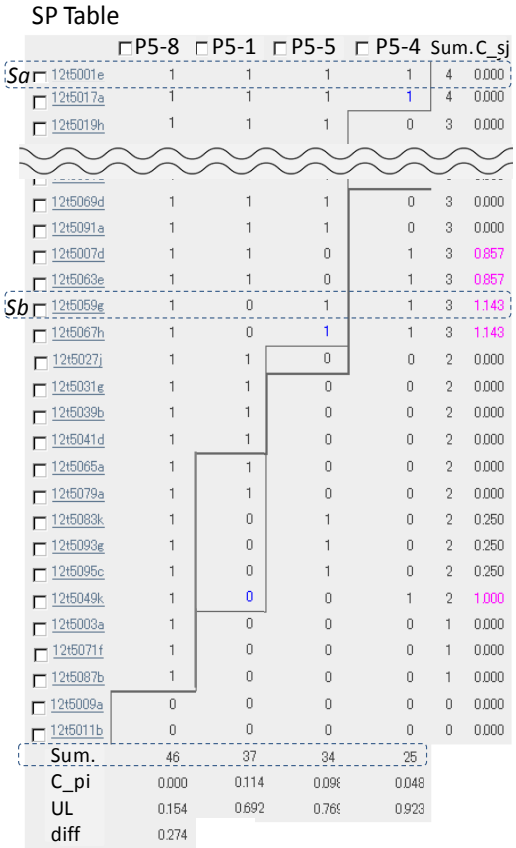


Fig. 6. A SP score table generated by our tool.

An example of an SP score table is shown in Fig. 6. The left column is the students' numbers. The top row is the problem IDs. A student whose ID is 12T5001e (marked as Sa) answered all the problems correctly, so all entries are 1. 12T5059g (Sb) answered the second problem (P5-1) incorrectly, so this entry is 0.

This table is sorted by column and by row in decreasing order of occurrence of 1s. The number of correct answers for problem P5-8 is 46. This is the best score of these four problems, so, this is located in the left most position. On the other hand, problem P5-4 is located in the right most position, because the number of correct answers is the worst of all the questions.

The students are sorted by their total score. If some students have the same score, they are ordered based on the sum of the number of correct answers to the problem which they answered correctly. For example, in the case of the student marked "Sb" and the student located above him, these students got 3 points. However, the Sb student answered all problems except for the second problem (P5-1) correctly. The sum of correct answers for the problems he answered correctly is 105 (46+34+25). On the other hand, the student above him answered all problems except for the third problem (P5-5) correctly. The sum for this student is 108 (46+37+25). This student has a higher sum, so he is located higher than the Sb student, even though they have the same total score.

As a consequence, the upper-left triangular region is filled with nearly all 1s. Ideally, students with higher scores should solve those problems which are answered correctly by most students. Similarly, if a problem is solved by most of the students, a good student is able to solve the problem.

C. Educational Practice

At the university level, 100 students/year/university in algorithmic thinking courses for freshmen from 2008 in 3 universities have used this system. At the high school level, 45 students/class/school in an elective subject named "Information Studies" from 2010 in 2 high schools have used our system.

The learning steps in these classes were calculation, selection and repetition in this order. In each step, first, teachers showed a sample algorithm with a template in order for the students to see and dissect the whole process, multiple block processes, and each block process before they described their algorithm. Second, teachers gave them an incomplete algorithm template that has blanks which must be filled in. An algorithm with some errors was also given to students, who had to find the bugs and correct them. Finally, students tried to build the whole algorithm from scratch.

From the evaluations from our users, the effects of our system are summarized as follows:

[For learners]

- Understanding about nested structures and the block-range of flow structures was fostered.
- The tracing accuracy of variables was improved.
- Each learner's score in the programming course was improved.

[For teachers]

- The evaluation time for learners' answers was decreased.
- The quantitative analysis of the individual missteps of each learner could be provided.

The pedagogical features of our system helped teachers correct learners' mistakes and point them out before programming. Additionally, this made teachers quantify whether learners' calculations were exhaustive, did not double count, and were efficient. Teachers could also understand in detail each learner's learning progress. The above could become one of the factors in improving the quality of education.

IV. Analyzing Results Using the SP Score Table

A. S Line & P Line

An example of an SP score table generated by our system is shown in Fig. 6. There are two types of line, a thin line and a bold line. The thin line is the S line and the bold line is the P line.

The S line is defined based on the scores of all students. Starting from the top student of the table, for each student, a short vertical line is drawn on the right of the score for the problem column which is the same as the student's score. For example, the Sb student's score is 3, so there is a line on the right side of third problem column. The S line shows the frequency distribution of the students' scores. Ideally, there should only be 1s on the left side on this line and 0s on the right side. Teachers can capture the status of the achievement of each student.

The P line is defined based on the score of all problems. From the most high score problem, for each problem, a horizontal line is drawn below the score for the student row which is the same as the number of correct answers for the problem This line shows the frequency distribution of the problems' score. For example, there were 46 correct answers for problem P5-8, so the line is drawn below student 12t5087b who is 46th in the table. Ideally, only 1s should be above this line and 0s below it. Teachers can capture the suitableness of each problem or the effect of his/her instruction.

These two lines are ideally located close together in the table. The difference between the S line and the P line shows some pedagogical agenda to be solved.

1) The suitability of problems for this learner group.
2) The effectiveness of the teacher's instruction for this learner group.
3) The achievement level of each student.
4) The variation in learning abilities of the learners.

B. Caution Indices

This method includes two index types: a caution index for students (C_s) and a caution index for problems (C_p) [12], [13].

1) Caution index for students

By using the C_s index, we can easily detect the students who need attention. Theoretically, the threshold value is 0.5. If there are students whose values are under 0.5, they show abnormal response patterns for the problems, i.e., careless mistakes by high achieving learners and lucky guesses by low achieving learners. Moreover, teachers can diagnose students who cannot use the learning environment, especially the description method.

The combination of the C_s index and the percentage of questions answered correctly expresses the achievement level of students. We call this graph the Student Score Graph. Fig. 7 shows our interpretation of the Student Score Graph. The vertical axis is the number of problems answered correctly

(Num. for short) and the horizontal axis is the C_s index. This graph is separated into 4 sections by the vertical axis (Num.) threshold value of 50% of total problems and the horizontal axis (C_s) threshold value of 0.5.

The left part of this graph is when the C_s index is less than 0.5. The students in this area seem to have normal reactions to the problems. If a student is in the upper left section (Num.>=50%), he/she is a "good" student. The lower left section (Num.<50%) is a "developing" student who needs more practice.

The right part is when the C_s index is more than 0.5. The students in this area seem to have abnormal reactions to the problems. Students in the upper right section, where Num. is high, tend to make careless mistakes for fundamental problems or to not fully acquire the related knowledge. When Num. is low and the C_s index is high, the comprehension of the students is quite uncertain. Students in the lower right section tend to have unstable answers. Therefore, they require special attention from the teacher.

In our research, the C_s index has been used to identify students who need attention in each lesson. In Fig. 6, there are five students whose C_s Indices are higher than 0.5. 12t5007d and 12t5063e are 0.857, 12t5059g and 12t5067h are 1.143 and 12t5049k is 1.00. These students answered incorrectly some problems which they were expected to answer correctly.

The right graph in Fig. 9 is an example of the Student Score Graph. The radius of each circle represents the number of students located at that position. The center position of each circle shows the score combination. The five students mentioned above are located in the lower right section. The teacher may give these students particular attention.

2) Caution index for problems

By using the C_p index, teachers can analyze the quality of a problem. We can detect inadequate questions which may include vague instructions or ill-structured tasks. Theoretically, the threshold value is 0.5. If C_p<0.5, teachers can modify or delete defective problems, and fit good ones into an item bank for future use. In Fig. 6, there are no problems that need to be improved because their C_p Indices are all under 0.5.

The combination of the C_p index and the percentage of problems answered correctly expresses the appropriateness of the problem to evaluate the students' abilities. We call this graph the Problem Score Graph. Fig. 8 shows our interpretation of the problem score graph. The vertical axis is the percentage of problems answered correctly (% in short) and the horizontal axis is the C_p index. This graph is separated in 4 sections by a % threshold value of 50% and a C_p threshold value of 0.5.

The left part of this graph is when the C_p index is less than 0.5. This means the problems in this area are "suitable" to evaluate students' abilities. If a problem is in the upper left section (%>=50), this is an "easy" problem. A problem in the lower left section (%<50) is a "difficult" problem which good students tend to answered incorrectly. The right part is when the C_p index is more than 0.5. This means the problems in this area are "unsuitable" problems to evaluate students' abilities. If a problem is in the upper right section, however, the % is high, so this problem includes some "different" factor to evaluate target abilities. A problem in the lower right section is a "vague" problem. The % is low and the C_p index is high. This means the problem includes ambiguity factors in its description. For example, the problem statement is difficult to read, or the way of answering the problem is not suitable for the learning environment.

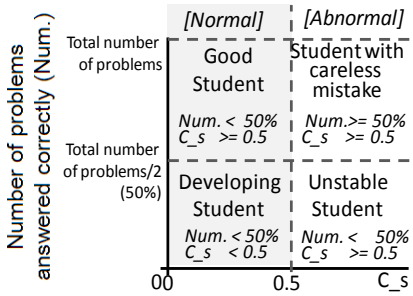


Fig. 7. The relation of the C_s index and the number of the problems answered correctly.

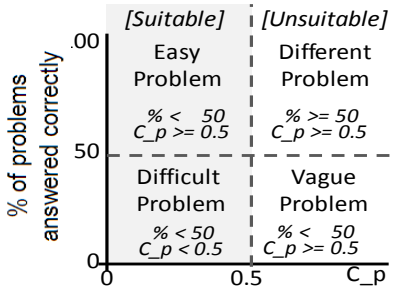


Fig. 8. The relation of the C_p index and the percentage of problems answered correctly.

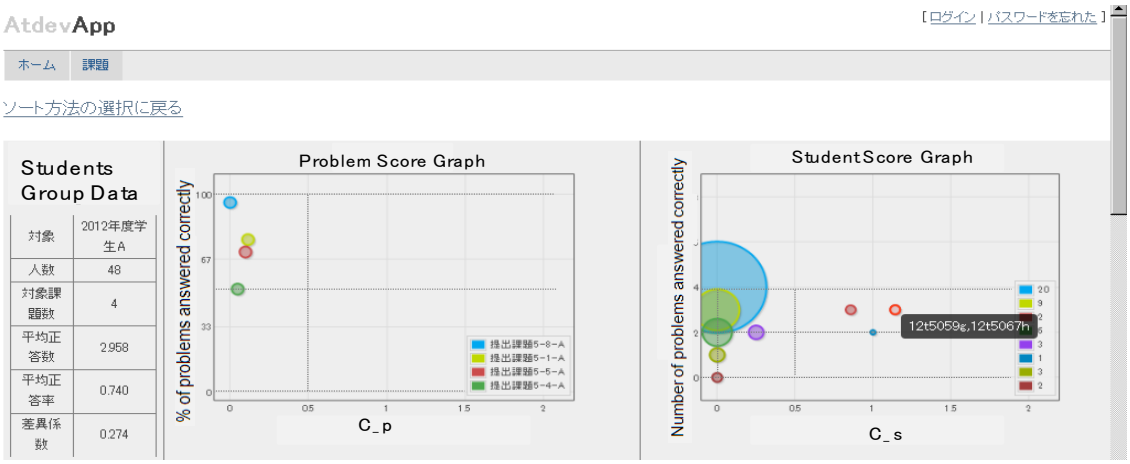


Fig. 9. Problem score graph and Student score graph generated by proposed our tool.

The C_p index has been used for evaluating the quality of each problem for each school year. The left graph in Fig. 9 is a Problem Score Graph. The four problems in Fig. 6 (P5-8, P5-1, P5-5 and P5-4) are plotted on this graph. They are all "suitable" problems. Three problems are located in the "easy" section and one problem is in the "difficult" section.

V. Conclusion

In this paper, we described the pedagogical features and main functions of a learning support system for algorithmic thinking education. We have used this system since 2008 in our computer science course. The ability to control the usage of algorithmic elements such as input, output, selection or repetition is a unique function that no other system has had before. Teachers can define the element usage rule depending on the learning progress of their learners and their educational philosophy. In the future, we will pedagogically analyze and design learning scenarios with this supporting system.

References

[1] H. S. Stone, *Introduction to Computer Organization and Data Structures*, New York: McGraw-Hill, 1972.

[2] D. E. Knuth, "Algorithms in modern mathematics and computer science," Stanford Department of Computer Science Report, 1980.

[3] N. Sarawagi, "A general education course - introduction to algorithmic thinking - using visual logic," *Journal of Computing in Sciences in Colleges*, vol. 25, no. 6, pp. 250-252, 2010.

[4] S. Hubalovsky, E. Milkova, and P. Prazak, "Modeling of a real situation as a method of the algorithmic thinking development and recursively given sequences," *WSEAS Transactions on Information Science and Applications*, vol. 7, no. 8, pp.1090 – 1100, 2010.

[5] G. Futschek and J. Moschitz, "Learning algorithmic thinking with tangible objects eases transition to computer programming," in *Proc. 5th International Conference on Informatics in Secondary Schools: Evolution and Perspectives*, 2011, pp. 155–163.

[6] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick, "Scratch: a sneak preview," in *Proc. of Second International Conference on Creating, Connecting and Collaborating through Computing*, 2004, pp. 104–109.

[7] B. Harvey and J. Monig. Bringing no ceiling to scratch: can one language serve kids and computer scientists? [Online]. Available: http://www.eecs.berkeley.edu/»bh/BYOB.pdf

[8] S. Cooper, W. Dann, and R. Pausch. Information systems education conference 2000. [Online]. Available: ftp://software.org.mx/docs/Karel/isecon00.pdf

[9] Y. Fuwa, Y. Kunimune, M. Kayama, M. Niimura, and H. Miyao, "Implementation and evaluation of education for developing algorithmic thinking for students in computer science," *IEICE Technical Report of Education Technology*, vol. 109, no. 268, pp. 51–56, 2009.

[10] J. Bruner, *The Process of Education*, MA: Harvard University Press, 1960.

[11] M. E. Cook, *Guided Discovery Tutoring: A Framework for Icai Research*, NY: Van Nostrand Reinhold, 1991.

[12] T. Satoh, "A classroom information system for teachers, with foocus on the instructional data collection and analysis," in *Proc. ACM '74 Annual Conference*, 1974, vol. 1, pp. 199-206.

[13] D. L. Harnisch and R. L. Linn, *Identification of Aberrant Response Patterns: Final Report*, Education Commission of the States, 1981.

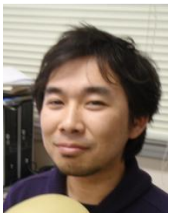
Mizue Kayama is an associate professor at Department of Computer Science and Engineering, Faculty of Engineering, Shinshu University in Japan from 2007. She obtained her PhD degree from the University of Electro-Communications in 2000. Her research fields are computer science education, artificial intelligence in education and knowledge management framework at educational context.

Makoto Satoh is currently a master course student at the Graduate School of Science & Technology, Shinshu University. He received B.E. in 2012. His research field is learning technology, especially for algorithm and algorithmic thinking education.

Kei Kabayashi is currently a master course student at the Graduate School of Science & Technology, Shinshu University. He received B.E. in 2012. His research field is learning technology, especially for algorithm and algorithmic thinking education.

Hisayoshi Kunimune is an assistant professor at Department of Computer Science and Engineering, Faculty of Engineering, Shinshu University. He has received his B.E., M.E., and Ph.D. degrees from Shinshu University, Nagano, Japan in 1998, 2000, and 2003, respectively. He was a postdoctoral fellow at the Faculty of Engineering, Shinshu University from 2003 to 2004. His research areas include learning and education supporting systems.

Masami Hashimoto is an associate professor at Department of Computer Science and Engineering, Faculty of Engineering, Shinshu University. He received B.E. degree in 1985 and Dr. Eng. degree in 2005 from Shinshu University. His research interests are human computer interaction and biomedical engineering



Makoto Otani is an associate professor at Department of Computer Science and Engineering, Faculty of Engineering, Shinshu University, Japan from 2011. He received Ph.D. degree from Kyoto University in 2006. In 2012, he was a visiting scientist at Audio & Acoustic Group, LIMSI-CNRS, France. His research interests are in Acoustical Engineering, especially spatial hearing and its application to spatial audio techniques.