Implementation of a GA Driven Programming Training Support System

Eiji Nunohiro, Kenneth J.Mackin, Masanori Ohshiro, Kotaro Matsushita, Kazuko Yamasaki
Department of Information Systems, Tokyo University of Information Sciences
1200-2 Yatoh-cho, Wakaba-ku, Chiba 265-8501, Japan
{nunohiro, mackin, ohshiro, matsushita, yamasaki}@rsch.tuis.ac.jp

Abstract

Based on the analogy that the process for building a program is similar to configuring a puzzle, we propose a programming training method in which the user reconfigures program fragments or program puzzle pieces, and improves program structure and program flow comprehension skills. The proposed training system applies genetic algorithm (GA) to realize the problem creation feature which creates a program puzzle which best matches the user's level of comprehension. The GA process evaluates both the level of comprehension of the user and the estimated difficulty of the puzzle, to determine the target problem difficulty, i.e. the difficulty of the puzzle. With this GA feature, we developed a programming training support system which automatically creates appropriate programming training problems matching the user's comprehension level. In this paper, we first describe the features and functions of the developed puzzle style programming training support system. Next, we describe the results of implementing the developed system to a C language programming course at Tokyo University of Information Sciences. Finally, we apply statistical tests to the results of changes in the students programming skills, to show the validity of the proposed system.

key words: genetic algorithm, e-leaning, programming education, T test.

1 introduction

Computer education, such as Information literacy and Programming training, has received much interest recently in a wide range of academic programs, including high schools and liberal arts undergraduate programs. In the case of early programming training, there are many topics that need to be studied together, i.e. programming language syntax, programming design methods for object oriented design, and problem solving algorithms.

In this research we aim at creating a training support system which targets an introductory or beginner level programming user, and allows the user to develop and improve programming skills through a game-like learning process.

There has been previous research on automatic generation of programming exercises [1]. In this system proposed by Suganuma et. al., in order to modify the difficulty of the created exercise, various parameters must be modified by the administrator. On the other hand, our proposed system determines the difficulty of the exercise by evaluating the current level of the user from the user's training history, and automatically creates appropriate programming exercises matching the user's level of understanding. The programming training support system developed in this research is designed upon the similarity between the building process of a software program and a jig-saw puzzle. The proposed system trains the programming skill of the user by providing exercises in which the user must rearrange source code fragments, similar to a puzzle building task. By providing an interface to solve programming exercises much like puzzle solving, it allows users who are weak in programming to continue training through a game-like experience. In our proposed system, the difficulty of the problem is adjusting by modifying the granularity of the source code fragments, and the location of the breakpoints of the fragments or puzzle pieces. For this system, we developed an algorithm applying genetic algorithm (GA) to automatically generate exercises of appropriate difficulty that match the user's level of proficiency[2].

We used the proposed training system in a beginner level class for a C language programming course in Tokyo University of Information Sciences. We analyzed the changes in the students' programming skill in order to evaluate the validity and effectiveness of the proposed system.

2 Outline of Programming Training Support System

In order to improve programming skills, it is important to practice coding programs to fulfill a given problem specifications. But it is also important to learn to analyze a given program code to (a) understand the algorithm flow, (b) confirm that the source code correctly fulfills the requirements, and (c) suggest possible improvements in performance or maintenance. The proposed programming training support system provides a training method to improve (1) program flow analyzing skills, and (2) program structures comprehension skills. The proposed system trains these aspects by having the user reconstruct programs from program fragment pieces, much like building a puzzle by finding the right puzzle pieces. The proposed system evaluates the student's current level of progress from past training history, and automatically generates programming exercises which match the student's level of progress.

The programming exercise is automatically generated using the following 2 features.

(a) The progress management feature calculates the student's current level of progress from the user's history of training with the system.

(b) The puzzle creation feature generates the program puzzle exercise from a completed program source code. The puzzle creation feature can create different patterns of puzzle exercises from the same source code, so that the user can repeatedly try solving different exercises with the same correct answer. The different patterns of puzzle exercises generated are selected to be of the correct level of difficulty for the user's current level of progress. The level of difficulty of the generated puzzle is determined by the number fragments and location of break points. Genetic algorithm is applied in the algorithm to determine the number of fragments, and the location of the break points to divide the program into fragments.

Processing flow of programming training support system is described in the following.

(1) System shows a problem to user according to the learning contents such as a syllabus of program practice or algorithm that user wishes.

(2) User selects a problem to answer from the problem that system showed

(3) System breaks up a program into puzzle pieces depending on user level by using genetic algorithm to determine the location and number of separation for the program.

(4) User reconstructs the program by selecting the correct program puzzle pieces in the correct order.

(5) System estimates user's result, and accumulates as the user history.

The basic component of this system to achieve the above mentioned processing is shown in Fig.1.

① **exercise display section** indicates the problem which is proposed by system using the learning item and the user's proficiency.

② **puzzle generation section** breaks up a program into puzzle pieces depending on user progress level by using genetic algorithm.

③ **exercise selection section** selects a problem from the learning item that user whishes or user proficiency.

④ **program management section** manages the program used as a problem.

⑤ **progress display section** indicates user progress status from user proficiency.

⑥ **progress management section** manages user progress status.

⑦ **progress evaluation section** evaluates user progress from the user proficiency.

⑧ **result display section** indicates a result of user.

⑨ **answer evaluation section** estimates user's answer.

3 Programming problem creation

In this research, we created programming exercises targeted towards beginner learners. The exercise required the student to arrange program fragments or pieces in the correct order to complete the program described. In general, if the exercise given is much too difficult or too easy for the student, the exercise will not assist the learning process of the student, and may adversely affect the motivation to study the subject. Therefore, it is necessary to create program puzzle exercises of appropriate difficulty according to the level of comprehension of the student. To realize this, we considered the following method to create program puzzle exercises.

【step1】 Analyze the program source code by determining for each program statement, a) the control

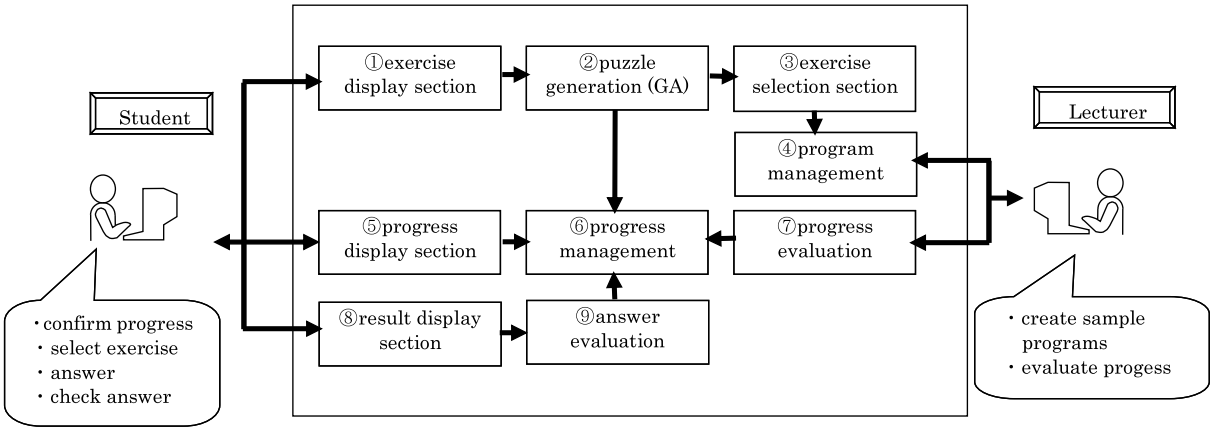

Figure 1: Configuration of programming training support system

structure depth (control information) and b) variable reference (reference information).

【step2】Calculate the difficulty level for each statement using the above control and reference information.

【step3】Apply genetic algorithm (GA) search to find the optimal combination of partition-points which best match the progress level of the user.

In the following sections, we describe the genetic algorithm search applied.

3.1 Puzzle fragment granularity

Possible granularity of program fragments or divisions are at identifier, expression, statements, or block levels. For this research we create program fragments between lines of code. Program fragment granularity at the expression level will be implemented in the future.

3.2 Source code analysis

Each line of code is analyzed for information to be used in determining the difficulty of the partition. For each line of code, the control depth, and variable reference is calculated to measure the complexity of the source code.

【step a】 control information analysis
The "control depth" for each statement is calculated. The control depth is the depth of the nest of control statements, such as if, for, and while statements.

```
                Program                                  Depth    Reference
 1:public class EvenOrOdd {                               0          0
 2:   public static void main(String[] args) {            1       1(def:1,use:0)
 3:     int Evev = 0;                                     1       1(def:1,use:0)
 4:     int Odd = 0;                                      1       1(def:1,use:0)
 5:     for(int i=0 ; i<=100 ; i++){                      1       4(def:2,use:2)
 6:       if(i % 2 == 0){                                 2       1(def:0,use:1)
 7:         Even = Evev+1;                                3       2(def:0,use:1)
 8:       }else{                                          2          0
 9:         Odd = Odd+1;                                  3       2(def:1,use:1)
10:       }                                               2          0
11:     }                                                 1          0
12:     System.out.println(" Number of Even : "+Even);    1       1(def:0,use:1)
13:     System.out.println(" Number of Odd"+Odd);         1       1(def:0,use:1)
14:   }                                                   0          0
15:}                                                      0          0
```

Figure 2: Example of program

【step b】 reference information analysis
The "variable reference" count for each statement is calculated. The following criteria are used to determine the variable reference for statement S:
- a variable is defined in statement S
- a variable is used in statement S
The variable reference count for a statement is the total number of times a variable is defined or used in the statement.

An example of source code analysis is shown in Fig. 2.

3.3 Relation between Program partition points and difficulty of puzzle

The program puzzle pieces are created by selecting partition points between two lines of code. The difficulty of the puzzle is dependent on the number of pieces and the location of the partition points. If a

partition occurs at a location with high control depth and high reference count, the puzzle can be assumed to have a high difficulty. From this assumption, we define the following function to evaluate the difficulty of the partition. Partition point difficulty k is the evaluated difficulty when a partition occurs after program code statement k. Partition point difficulty Ppd_k is defined as:

$$Ppd_k = control\_depth * reference\_count \quad (1)$$

The difficulty of the puzzle is also affected by the difficulty of the program algorithm used in the given program source code, but this degree difficulty is dependent on the prior knowledge of the individual user. For this research we let the user decide on the degree of difficulty for an algorithm.

3.4 Program partition by GA

The partition pattern (number of partitions and location of partitions) is selected using the user's progress level and puzzle difficulty. This puzzle difficulty is calculated using algorithm and language specification that is used in problem and partition difficulty which is total value of partition point difficulty for each partition point. We apply genetic algorithm (GA) to select the partition pattern.

3.5 Chromosome expression

The partition pattern expressed as a binary string is used as the chromosome in the genetic algorithm. The length of the chromosome is 1 less than the lines of code (statements) in the program. In the chromosome, the value 1 indicates a partition at that location (line), and 0 indicates no partition at that location. Fig. 3 illustrates the relationship between chromosome and partition points. In Fig. 3 an example of a 15 line program is shown, and the partition points are the locations where the chromosome value is 1, i.e. after lines 2,5,6,8,10.

3.6 Fitness evaluation

Fitness is evaluated using the user's level of progress, target problem difficulty, and difficulty of the created program puzzle. Below we describe the fitness evaluation method.

【puzzle difficulty】
Puzzle difficulty is decided by 3 factors.

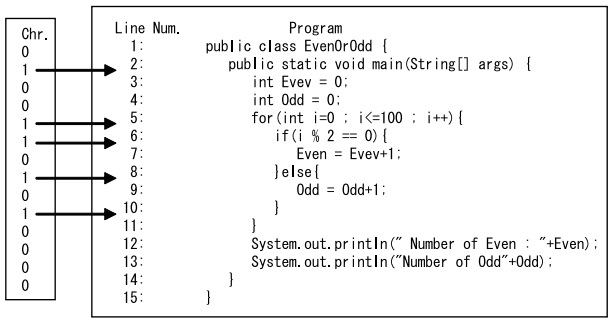

Figure 3: Example of program

① language specification which is being used by a program.

② algorithm which is used in a program.

③ partition difficulty for a program.

The above ① and ② are difficulty about a component of a program, and this these are divided into 10 stages as following.

* Level of language specification : $g(1 \leq g \leq 10)$
* Level of algorithm : $a(1 \leq a \leq 10)$

The partition difficulty is the sum of the partition point difficulty value for the given program puzzle chromosome, calculated with the following equation.

$$partition \;\; difficulty(pd) = \sum_{k=1}^{n} (Ppd_k * Chr_k) \quad (2)$$

where: n is number of program, Ppd_k is the puzzle difficulty at position k, Chr_k is the value of the chromosome at position k.

Puzzle difficulty is calculated as follows using the above g, a and pd.

$$puzzle \;\; difficulty(d) = g * a * pd \quad (3)$$

【User's level of progress】
The user's level of progress is defined as the puzzle difficulty of the last cleared problem.

【fitness function】
The value of the degree of adaptation is to estimate whether degree of difficulty of a problem is suitable for the intelligibility of the learner, and it's calculated using the degree of fitness function.

The fitness function is defined as the following:

$$fitness = |p - d| \quad (4)$$

where : p is user's level of progress, d is the puzzle difficulty.

The chromosome with the fitness value closest to 0 is selected as the optimal program puzzle combination for the user.

4 Training Implementation and System Evaluation

4.1 Course procedure and evaluation method

The proposed system was implemented in a beginner level class for a C language programming course for 1st year students in the environmental information department curriculum at Tokyo University of Information Sciences. In the programming course, all students take a preliminary questionnaire in order to divide the course in 4 classes depending on past programming experience and knowledge. The four classes are higher level, standard level, and 2 beginner level classes. The beginner level students are divided into 2 classes which are beginner class A and beginner class B from the questionnaire taken at the beginning of the course. Fig. 4 shows the course and evaluation flow of the beginner classes. The number in the figure represents the number of 90 min lectures. The questionnaire includes short quizzes which evaluate aptitude in logical thinking, and students with higher scores were placed in the beginner class A, and students with lower scores were placed in the beginner class B. As a result, 31 students were placed in beginner A, and 24 students were placed in beginner class B. The proposed training system was applied only to the beginner class B after the first mid-term exam, and changes in the exam scores between the mid-term and end-of-term exams were compared and evaluated.

4.2 Course Content

Table 1 shows the lecture content for the beginner class. Basic programming language topics were explained through lecture, and students create simple programs applying the selected topic. The contents of the lecture were identical for both beginner class A and beginner class B, and the educational experience of the lecturers for both classes were roughly the same.

4.3 Evaluation Results

Table 2 shows the results of the mid-term exam, the end-of-term exam, and the statistical T test. From the results of the mid-term exam, there was a large difference between the average scores of the 2 beginner classes, and at 5% significance level, a statistical significance was noted between beginner class A and beginner class B. From the results of the end-of-term exam, the difference in average scores between the 2 classes is much smaller, and a statistical significance was not found between beginner class A and beginner class B.

In order to collect the subjective views of the students towards the proposed system, an anonymous questionnaire (5 being the highest rating) was taken after the end-of-term exam. Table 3 shows the results

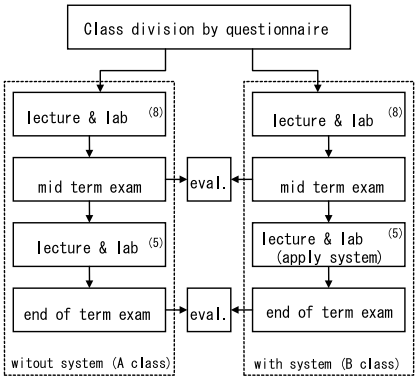

Figure 4: Course and Evaluation Flow

Table 1: Lecture content

lecture topic
- constants and variables
• I/O statements (printf, scanf)
• if statement (include nesting)
• for statement (include nesting)
• array (1 dimensional array)

Table 2: Exam and T Test results

class		mid	end
without system beginner A	average	73.7	66.3
	standard deviation	20.3	19.6
with system beginner B	average	61.9	63.3
	standard deviation	23.0	20.3
t test of beginner A & beginner B	t value	2.02	0.56
	significance	*p<0.05	none

Table 3: Questionnaire results

question	results[%]				
	5	4	3	2	1
1. The system was effective in improving programming skill.	18.2	59.1	22.7	0.0	0.0
2. The system will be useful in future studies.	30.4	52.2	17.4	0.0	0.0

of the anonymous questionnaire.

The questionnaire also asked for free opinions regarding the system in general. For positive views, comments such as "It was fun to study using the system." and "It was easy to understand programming using the system" were received. On the other hand comments such as "The operability was poor" gave feedback on points to be considered in the future.

4.4 Discussion of Results

From the exam results in Table 2, it can be seen that the beginner class B which had much lower average scores at mid-term compared to beginner class A, had improved to similar scores by the end of term. From Table 3 showing the results of the questionnaire, 77.3% of the students marked 4 or higher to the question "The system was effective in improving programming skill". Also, 82.6% of the students marked 4 or higher to the question "The system will be useful in future studies". From the free opinion question, many positive comments such as "It was fun to study using the system" and "It was easy to understand programming using the system" were received. From these results, we conclude that the proposed training system had been effective in supporting programming education.

On the other hand, points to be considered in the future were raised through the questionnaire. There were comments on the poor operability of the system, and we feel that the man-machine interface and screen layout needs to be reconsidered for improvement.

5 Conclusion

For software programming classes and other lab based courses, there is an inherent problem of how to address the needs of the beginner and slower learning students. Especially when the size of the class becomes larger, it becomes more difficult to supervise every student individually. The effectiveness of the training course can be improved by providing a self learning environment in which these students can study at their own paces.

In this research, we proposed a programming training support system which targeted beginner and slower learners. The system aims at improving programming skill through a game-like interface. We applied the proposed system in a C language course, and evaluated the educational effectiveness of the system. From the research results, we were able to show that this system was effective in supporting the learning of beginner and slower students. We feel that similar positive results can be expected in programming courses for students without strong mathematical background.

For future works, we plan to redesign the man-machine interface and screen layout to improve the operability of the system. We also plan to add features to enable algorithm training and object-oriented programming training, in order to expand the application area of the proposed training system.

References

[1] AKIRA SUGANUMA, TSUNENORI MINE, TAKAYOSHI SHOUDA, Automatic Exercise Generating System That Dynamically Evaluates both Students 'and Questions 'Levels, IPSJ(Information Processing Society of Japan) Journal, pp.1810-1817, Vol.46 No.7 July 2005

[2] Eiji Nunohiro, Kenneth J. Mackin, Masanori Ohshiro, Kazuko Yamasaki, Applying Genetic Algorithm to a Programming Training Support System, The eleventh International Symposium on Artificial Life and Robotics(AROB 11th '06) Program p75, Proceeding Index GS10-1, January 2006