Procedure Generation for Algorithm Learning System using Comment Synthesis and LSTM

Akiyoshi Takahashi *, Hiromitsu Shiina †, Ryunosuke Ito ‡, Nobuyuki Kobayashi §

Abstract

We have constructed a learning system by organizing procedures for learning program creation. However, manually creating procedures for learning systems requires a significant amount of time. In this study, in addition to automatically generating program procedures using natural language processing, we generate new program content and procedures by learning program code and comments through deep learning long short-term memory.

Keywords: Programming learning, Comment Generating, Summarization, Neural Machine Translation, Encoder–Decoder Translation Model

1 Introduction

Training the ability to utilize information is important for the progress of an information society. In school education, it is important to enrich information and communications technology (ICT) education at every stage. Preparation has already begun in elementary, middle, and high schools; for example, programming education in elementary schools will be compulsory by 2020 in Japan [1]. Considering programming education for university students and the connection to ICT education in elementary, middle, and high school, it is necessary to prepare content for programming education and study its support system. In particular, it is important in ICT education to operate programs directly on computers; however, learning computational thinking [2] is more important than learning programming grammars. Many studies on programming education have been done [3]. As a related study, there is research on programming education for beginners using operation logs [4, 5].

In computational thinking, we have discussion on the relationship between computational thinking and logical thinking. Relationship with language management is being debated [6]. Though, the purpose of this study is to aid in understanding the procedures for solving problems, we have constructed an algorithm learning system that rearranges algorithm procedures on a tablet PC [7]. This system is intended for students studying programming at the university level. It differs from learning programming language grammar

* Graduate School of Informatics, Okayama University of Science, Okayama, Japan
† Okayama University of Science, Okayama, Japan
‡ Advanced Information Design, Tokyo, Japan
§ Sanyo Gakuen University, Okayama, Japan
and aims to simplify learning algorithms. We have evaluated the use of this system by international students at Okayama University of Science in Japan. Creating many exercises requires time and effort on the part of the lecturer; therefore, a system for automatically generating these exercises is beneficial and necessary. As a related work in the field of software engineering research, there is research on the generation of nouns for programming comment generation [8]. However, it is not study on the generation of comments.

As the first step, a summarization technology [9, 10, 11] of division, merging, and paraphrase processing is used for program code comments. Since procedures are similar to comments, both can be generated automatically. The development of a method using neural networks has led to the development of natural language translation and sentence generation. In particular, an encoder–decoder translation using long short-term memory (LSTM [12]) has been proposed, which improved the translation accuracy. As the second step, pairs comprising one line of program code and its comments are learned in the encoder–decoder translation model used for the translation of program code to comments. In addition, LSTM generates a comment for a new piece of program code. In this study, we used 53 programs written in the C language used in first year university lectures in Japan. While learning the computational thinking, it is important to be able to summarize and disassemble the explanation of an algorithm. Although it is not necessary to understand variables in the first stage of learning, it is necessary to understand the programming language in the second stage. In particular, it is necessary to understand changes in variables. In addition, since information about the variables is insufficient from the program code originally used for learning, the flow relating to the program variables tends to be difficult to understand. Therefore, the comment generation process includes the added conversion of variable information of learning program code and variable names to generate comments by LSTM.

2 Algorithm learning system

Programming involves a number of procedures, such as declaring variables, inputting values, performing calculations, and outputting results. This programming procedure test involves converting programming into Japanese and dividing the procedures into several parts. There is text that rearranges the divided procedures in sequential order so that they match the flow of the programming. This text is utilized for understanding learning situations in lectures.

For example, we consider a foreign currency conversion problem that involves creating a program that takes an amount in yen as input and returns its conversion to US dollars, pounds, or euros as output. The procedure for solving this problem can be divided into sections, as shown in Figure 1(a). In this study, we have developed a system for testing the problem of rearranging this text, as shown in Figure 1(b).
3 Procedure generation through comment synthesis using program structures

In the algorithm learning system, procedures are generated by summarizing comments based on their vocabulary and degree of association with the lecture material and the program structure, as shown in Figure 2.
3.1 Evaluation of comment priority

Comment priority can be evaluated based on the following procedure. The overlap of important words between the comments in the program code and the lecture materials is given by the F-value, as shown in Figure 3.

(1) Obtain the term frequency inverse document frequency (TF-IDF) in the course material.
(2) Obtain the F-value for important words in the course material and program comments.
(3) Determine the comment evaluation value, which is the total F-value for the words included in the comment. This value selects comments with F-values over a certain threshold.

Figure 3: Important word shared by program comments and lecture material by F-value

3.2 Procedure generation using program structure

After deleting the comments, we merge the remaining comments using the merging conditions shown below. An example of the procedure generation is shown in Figure 4.

(1) For an unused function, the comments at its beginning and end are deleted.
(2) Parallel integration: comments at the same level in the program code in which the same noun is at the end of the statement are summarized as one statement.
(3) Deletion based on F-value: comments with small F-values are deleted.
(4) F-value priority: lower layer comments are preserved for comments with large F-values.

Figure 5 illustrates the comment synthesis results, using an example of source code.
### Figure 4: Extraction of important comments by F-value

<table>
<thead>
<tr>
<th>Level</th>
<th>Comment</th>
<th>Generated Procedure</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>最小値の定義</td>
<td>Definition of minimum value</td>
<td>並列の統合 Parallel integrations</td>
</tr>
<tr>
<td>1</td>
<td>最大値の定義</td>
<td>Definition of maximum value</td>
<td>並列の統合 Parallel integrations</td>
</tr>
<tr>
<td>1</td>
<td>配列の追加</td>
<td>Enter numerical value in array</td>
<td>F値の値選択 Selection by F value</td>
</tr>
<tr>
<td>2</td>
<td>整数変数の宣言</td>
<td>Declaration of integer type array variable</td>
<td>並列の統合 Parallel integrations</td>
</tr>
<tr>
<td>2</td>
<td>配列内の確認</td>
<td>Confirm the array</td>
<td>F値の値選択 Selection by F value</td>
</tr>
<tr>
<td>3</td>
<td>配列内の確認</td>
<td>Confirm the array</td>
<td>F値の値選択 Selection by F value</td>
</tr>
<tr>
<td>3</td>
<td>メイン関数の終了</td>
<td>End of main function</td>
<td>削除 Delete</td>
</tr>
</tbody>
</table>

### Figure 5: Comment synthesis results

**Program code (C programming language)**

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    int n, i, j;
    double a[100], max, min;
    for (i = 0; i < n; i++)
        scanf("%lf", &a[i]);
    max = a[0];
    min = a[0];
    for (i = 1; i < n; i++)
    {
        if (a[i] > max)
            max = a[i];
        if (a[i] < min)
            min = a[i];
    }
    printf("Maximum value: %f\n", max);
    printf("Minimum value: %f\n", min);
    return 0;
}
```

**Generated Comments**

1. Define the size, minimum value, maximum value of the array.
2. Name the variables.
3. Declare the Main function.
4. Enter numerical values in all subscripts of the array.
5. Case that is not within the range from the minimum value to the maximum value.
6. Error handling.
3.3 Procedure evaluation based on comment synthesis using program structure

The deletion rate of the program comments extracted for evaluation was approximately 60%. Additionally, nine participants studying the program participated in a questionnaire with a five-point rating system (ranging from 1 [bad] to 5 [good]) regarding the following two questions:

(1) Evaluation of procedure text
(2) Program understanding related to the procedure

The results of this questionnaire are shown in Table 1. In the text evaluation of the questionnaire, half or more of the participants evaluated it as ordinary text; however, when asked to evaluate their understanding of the content of the program based on this text, many participants awarded a rating of 2, which is considerably low. We conjectured that the large amount of low ratings resulted from the fact that the procedure generation ended with an initial declaration of variables.

<table>
<thead>
<tr>
<th>Questionnaire</th>
<th>Evaluation(1-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>(1) Evaluation of procedure</td>
<td>0%</td>
</tr>
<tr>
<td>(2) Program understanding related to</td>
<td>0%</td>
</tr>
<tr>
<td>the procedure</td>
<td></td>
</tr>
</tbody>
</table>

4 Procedure generation with LSTM

In cases where the program code contains comments, it is possible to generate procedures using the summary technology described in the previous section; however, not all program code has attached comments. Therefore, it is necessary to learn to generate comments on program code lacking comments from the programs with attached comments. Previous studies considered using an encoder–decoder translation model [13, 14] that use deep learning LSTM to translate. In addition, sentence summarization by neural network is proposed by Rush [15].

In this study, we use this translation model to translate the program code into comments and attempt to generate comments and procedures. There are two steps to the process. The first is the generation of procedures one line at a time from the program code. The second is the generation of procedures related to program block units across multiple lines, such as if and while constructs. Procedures pertaining to the program code are generated by learning using the encoder–decoder translation model via LSTM on the program code and comment pair discussed in the previous section. In addition, procedures are generated for programs that do not have attached comments. To generate procedures using the encoder–decoder translation model, it is necessary to perform preprocessing on the learned data, define the encoder–decoder translation model, and learn the data. The general flow of the procedure generation model is shown in Figure 6.
Figure 6: Encoder–decoder translation model for neural machine translation from comments to procedure

4.1 Preprocessing of learning data of program code and comments.

To learn using the encoder–decoder translation model, it is necessary to separate the program source code from the comments. The LSTM model takes a sequence of program source code tokens as input and a word sequence of comments as $X = x_1, \ldots, x_i, \ldots$. The token is converted to the token ID number, and similarly, the word is converted to the word ID number.

4.2 Encoder–Decoder translation model using LSTM

With LSTM blocks, a sigmoid layer is used, and the information that should and should not be considered is set to 0 and 1, respectively. Furthermore, the error from the data saved in advance for the translation pair is lost, and the loss for each LSTM block is accumulated. Finally, the parameters are learned by performing an error back propagation method on the loss.

1. With encoder–decoder translation model $\alpha_t$, a normalization of the degree of similarity is performed for output $h_t$ in relation to $y_t$, and $\tilde{h}_t$ on the encoder side. $c_t$ creates context vectors using the degree of similarity obtained with $\alpha_t$ and $\tilde{h}_t$.
2. In the LSTM output using the encoder–decoder translation model, $c_t$ and $\tilde{h}_t$ are linked to create a vector, and weight is applied with a linear operator. By applying the activation function tanh to this, the intermediate layer $\tilde{h}_t$ is output in relation to $y_t$. Finally, a weight is applied with a linear operator with respect to $\tilde{h}_t$, and $\tilde{y}_t$ is output with the softmax function.

4.3 Data learning with a source code comments pair

Learning is performed multiple times using the defined procedure generation model. LSTM is used for learning data, and screening occurs using the sigmoid layer to select whether input $x_i$ is required. Furthermore, the loss with the accuracy data that was saved in advance is obtained. The accumulated loss is obtained and the parameter is retained after performing...
an error back propagation method. The retained parameter is passed to the next model and learning occurs. Additionally, in the LSTM implementations and algorithms, there are models that generate procedures learned from the tokens and comments for each line and block of program code.

4.4 Procedure generation and evaluation

Examples of generating 20, 100, and 500 learning times for the model learning by line and by block are shown in Figures 7 and 8, respectively.

(1) Evaluation of procedures generated by line
For procedures generated by line, learning in terms of comments was not achieved at all after several learning attempts; however, nearly 20 learning attempts resulted in the desired procedures. After 100 learning attempts, procedures were properly generated in relation to approximately 80% of all of the program code. After 500 learning attempts, the procedure generation converged, but in the case of other procedures repeatedly generated in an unstable way, there was a decrease in the number of procedures generated with a correct meaning in Japanese.

(2) Evaluation of procedures generated by block
In contrast to procedures generated by line, procedures generated by block were generated after a smaller number of learning attempts. Those with simple and short blocks or a large portion of code similar to the learning data were able to generate good code. In contrast, the procedure generation text that failed either included many words that were not registered as data or lacked distinct terms. The results were also dependent on the volume of the learning data.

4.5 Rating using a questionnaire on procedure generation

The results of the manually conducted questionnaire are shown in Tables 2 and 3, and the following observations can be made based on the ratings.

(1) When rating by line, the participants often understood the meaning of the procedures. The procedures were considered easy to understand regardless of the flow of the program. (2) There was a significant difference in results depending on whether all blocks had either different or nearly identical learning data. This accounts for the clear differences in ratings indicating whether the procedures were understandable. However, there was a poor result because the meaning of the important sections of the processing was not understood.
Figure 7: Example of procedure generations

(a) Test program

1: #include <stdio.h>
2: #define MAX 80
3: int main(void){
4: char string[MAX] = {'0'};
5: int i;
6: 
7: printf("文字列");
8: scanf("%s", string);
9: printf("%sが入力されました。", string);
10: }

(b) 20 learning times

1: Include header
ヘッダーの読み込み
2: Define the maximum value
最大値を定義
3: Declare the integer type array
整数型の配列を宣言
4: For input string
入力文字列用
5: Declare an integer type variable
整数型変数を宣言
6: 
7: Display texts prompting input processing
入力処理を促す文章を表示
8: Make an integer
整数を作る
9: Display input character string
入力された文字列を表示
10: 

(c) 100 learning times

1: Include header
ヘッダーの読み込み
2: Define the maximum value
最大值を定義
3: Declaration of main function
main関数の宣言
4: For input string
入力文字列用
5: Declare an integer type variable
整数型変数を宣言
6: 
7: Display texts prompting input processing
入力処理を促す文章を表示
8: Enter character string
文字列を入力
9: Display the input character string
入力された文字列を表示
10: 

(d) 500 learning times

1: Include header
ヘッダーの読み込み
2: Define the maximum value
最大値を定義
3: Declaration of main function
main関数の宣言
4: For input string
入力文字列用
5: Declaration of integer variable
整数型変数の宣言
6: 
7: Display sentence urging input processing
入力処理を促す文章を表示
8: Enter character string
文字列を入力
9: Display text to prompt input processing
入力処理を促す文章を表示
10: 

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.
Figure 8: Procedure generation for each block of source code by encoder–decoder translation model

Table 2: Evaluation of generated procedure by line

<table>
<thead>
<tr>
<th>Line number</th>
<th>Generated procedure</th>
<th>Evaluation(1-6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>1</td>
<td>Include standard input output file</td>
<td>0 0 0 0 0 5</td>
</tr>
<tr>
<td>2</td>
<td>Definition of maximum value</td>
<td>0 0 0 0 0 5</td>
</tr>
<tr>
<td>3</td>
<td>Declaration of main main's declaration</td>
<td>0 0 0 0 0 5</td>
</tr>
<tr>
<td>4</td>
<td>For input string</td>
<td>0 0 3 2 0 1</td>
</tr>
<tr>
<td>5</td>
<td>Declare the integer type variable</td>
<td>0 0 0 0 0 5</td>
</tr>
<tr>
<td>6</td>
<td>Display text prompting input processing</td>
<td>0 0 0 0 0 5</td>
</tr>
<tr>
<td>7</td>
<td>Enter a character string</td>
<td>0 0 0 0 0 5</td>
</tr>
<tr>
<td>8</td>
<td>Display the input character string</td>
<td>0 0 0 0 0 5</td>
</tr>
</tbody>
</table>

Copyright © by IIAI. Unauthorized reproduction of this article is prohibited.
Table 3: Evaluation of generated procedure by each block

<table>
<thead>
<tr>
<th>Line number</th>
<th>Generated procedure</th>
<th>Evaluation(1-6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>1–5</td>
<td>none</td>
<td>1 0 0 1 0 3</td>
</tr>
</tbody>
</table>
| 7–9         | input processing  
入力処理 | 0 0 0 0 0 5    |
| 11–17       | Output of outliers  
異常値の出力 | 0 5 0 0 0 0    |
| 19–21       | Output of result  
結果の出力 | 0 0 3 2 0 0    |

5  Procedure generation with addition of variable information

In computational thinking learning, it is necessary to be able to summarize and disassemble the explanation of an algorithm. Although it is not necessary to understand variables in the first stage of learning, it is necessary to understand the programming language in the second stage. In particular, it is necessary to know about variable changes. In addition, since information about the variables is insufficient with only the program code originally used for learning, the flow related to the program variables tends to be difficult to understand. Therefore, the procedure generation process is introduced in added conversion of the variable information of the learning program code and variable names to generate procedures by LSTM. The process of generating procedures including variables is explained separately for the conversion of learning data and the conversion of test data.

(1) Process learning data variables.
(1-1) Convert the learning program variables into temporary variable names, such as string1 for x(string1 → x) and i for x(i → x).
(1-2) Convert the learning data variable to a temporary variable name (x).
(1-3) Learn data that processes variable information repeatedly.
(2) Process test data variables.
(2-1) Obtain the variable information of each line from the test data to apply it to procedure generation.
(2-2) Convert the test data variable to a temporary variable name.
(2-3) Generate procedures from the test data whose variables are returned using LSTM.
(2-4) Restore the original variable name from the temporary name in the generated procedure.
Procedure Generation for Algorithm Learning System using Comment Synthesis and LSTM

Figure 9: Procedure generation by LSTM and procedure conversion after variable name processing

Figure 9 shows the procedure generation by LSTM, the comment conversion after variable name processing, correctness of generated procedure and correctness of addition of variable information by five students. We evaluated each line of the generated procedure. Two questionnaires were administered to five participants in the fourth grade, each questionnaire evaluating the validity of the program and the variable information. The evaluations of the procedure generation and variable information had high ratings, and the accuracy of complementing the variable information was also high.

6 Conclusion and future work

In this study, we generated procedures to simplify understanding algorithms. We used source code and comments as pairs of learning data. It is possible to generate procedures from a source code when the purpose of the program is limited, such as in the case of course material. Structure information is not using in learning data pairs. Using parse data as structure information is an remarkable improvement for generating procedures for blocking source code. In future research, we would like to extend the translation model beyond source codes and generate LSTM comments in a merged form, from both textbook and course information.
References


