IIAI International Journal of Service and Knowledge Management
International Institute of Applied Informatics
2019, Vol. 3, No. 2, 48 – 61

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 Shcool of Informatics, Okayama Univerity of Science, Okayama, Japan
† Okayama Univerity of Science, Okayama, Japan
‡ Advanced Information Design, Tokyo, Japan
§ Sanyo Gakuen University, Okayama, Japan

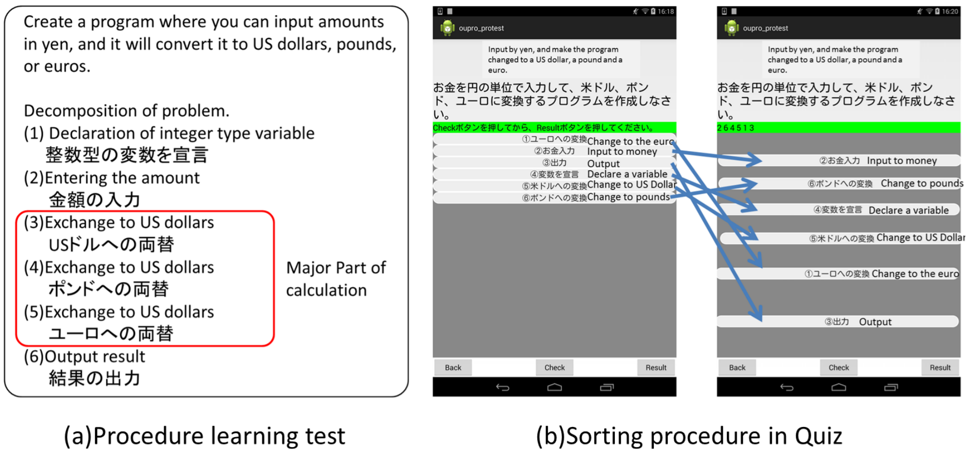
and aims to simplify learning algorithms. We have evaluated the use of this system by international students at Okayama University of Science in Japan. Creating many exercises requires time and effort on the part of the lecturer; therefore, a system for automatically generating these exercises is beneficial 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).



Create a program where you can input amounts in yen, and it will convert it to US dollars, pounds, or euros.

Decomposition of problem.
(1) Declaration of integer type variable
整数型の変数を宣言
(2)Entering the amount
金額の入力
(3)Exchange to US dollars
USドルへの両替
(4)Exchange to US dollars
ポンドへの両替
(5)Exchange to US dollars
ユーロへの両替
(6)Output result
結果の出力

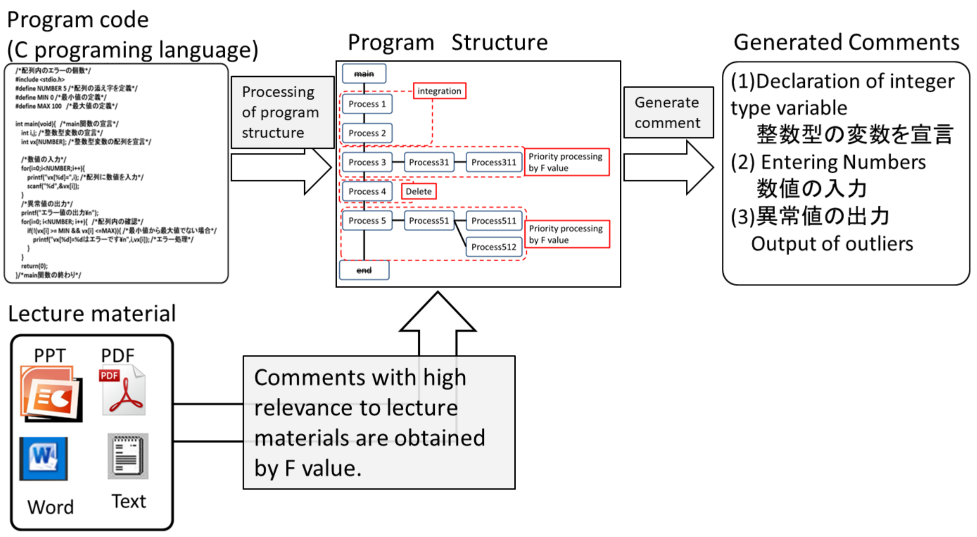
Major Part of calculation

oupro_protest
Input by yen, and make the program changed to a US dollar, a pound and a euro.
お金を円の単位で入力して、米ドル、ポンド、ユーロに変換するプログラムを作成しなさい。
Checkボタンを押してから、Resultボタンを押してください。
①ユーロへの変換 Change to the euro
②お金入力 Input to money
③出力 Output
④変数を宣言 Declare a variable
⑤米ドルへの変換 Change to US Dollar
⑥ポンドへの変換 Change to pounds
2 6 4 5 1 3
Back
Check
Result
(a)Procedure learning test
(b)Sorting procedure in Quiz

Figure 1: Procedure learning test and sorting procedures

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.



```
/*配列内のエラーの個数*/
#include <stdio.h>
#define NUMBER 5 /*配列の添え字を定義*/
#define MIN 0 /*最小値の定義*/
#define MAX 100   /*最大値の定義*/

int main(void){   /*main関数の宣言*/
   int i,j; /*整数型変数の宣言*/
   int vx[NUMBER]; /*整数型変数の配列を宣言*/

   /*数値の入力*/
   for(i=0;i<NUMBER;i++){
      printf("vx[%d]=",i); /*配列に数値を入力*/
      scanf("%d",&vx[i]);
   }
   /*異常値の出力*/
   printf("エラー値の出力¥n");
   for(i=0; i<NUMBER; i++){   /*配列内の確認*/
      if(!(vx[i] >= MIN && vx[i] <=MAX)){ /*最小値から最大値でない場合*/
         printf("vx[%d]=%dはエラーです¥n",i,vx[i]); /*エラー処理*/
      }
   }
   return(0);
}/*main関数の終わり*/
```

Generated Comments
(1)Declaration of integer type variable
整数型の変数を宣言
(2) Entering Numbers
数値の入力
(3)異常値の出力
Output of outliers

Figure 2: Outline of procedure generation system by comment Synthesis

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.

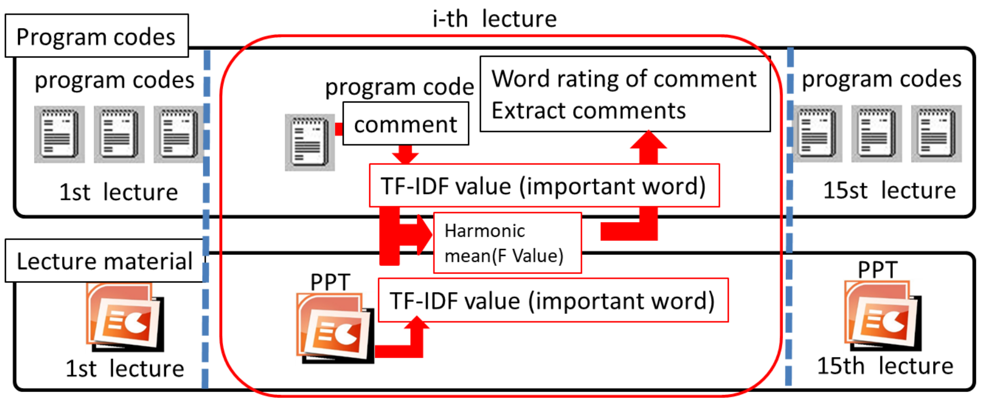
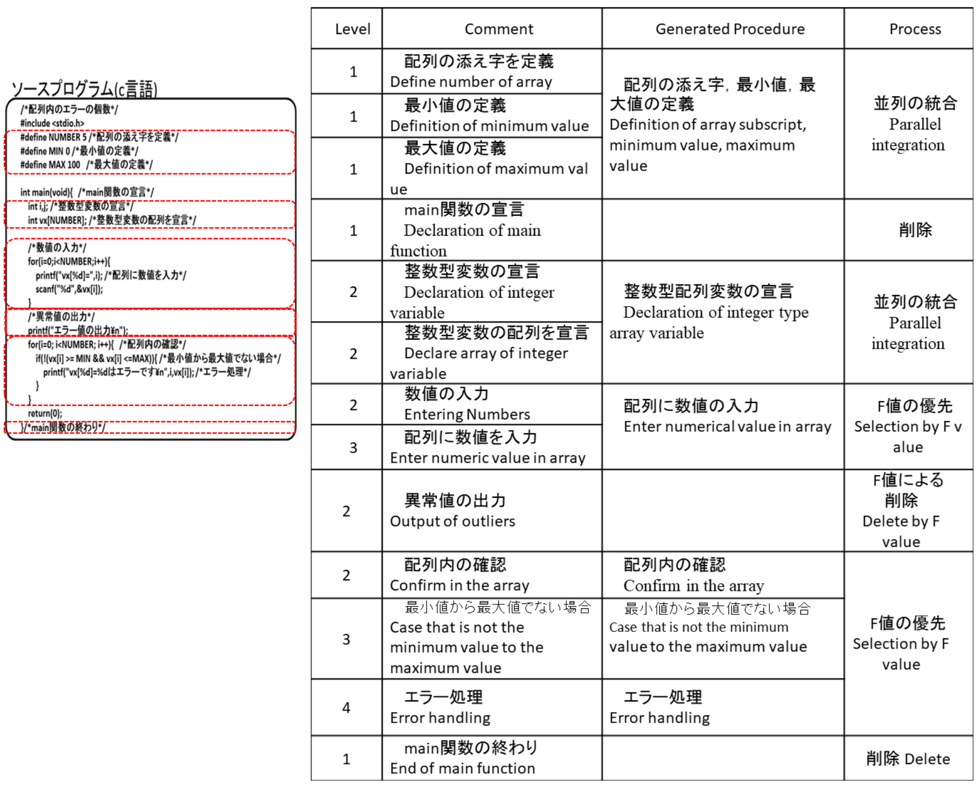


Figure 3: Important word shared by program comments and lecture material by F-value

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-value s 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.



ソースプログラム(c言語)

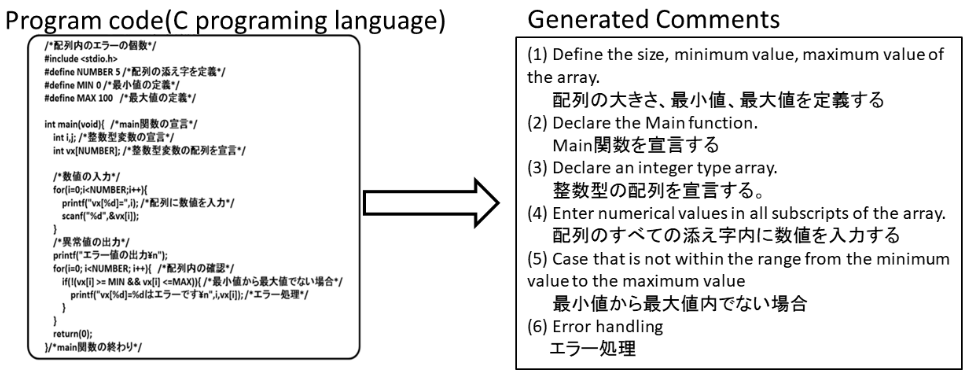
```
/*配列内のエラーの個数*/
#include <stdio.h>
#define NUMBER 5 /*配列の添え字を定義*/
#define MIN 0 /*最小値の定義*/
#define MAX 100   /*最大値の定義*/

int main(void){   /*main関数の宣言*/
   int i,j; /*整数型変数の宣言*/
   int vx[NUMBER]; /*整数型変数の配列を宣言*/

   /*数値の入力*/
   for(i=0;i<NUMBER;i++){
      printf("vx[%d]=",i); /*配列に数値を入力*/
      scanf("%d",&vx[i]);
   }
   /*異常値の出力*/
   printf("エラー値の出力¥n");
   for(i=0; i<NUMBER; i++){   /*配列内の確認*/
      if(!(vx[i] >= MIN && vx[i] <=MAX)){ /*最小値から最大値でない場合*/
         printf("vx[%d]=%dはエラーです¥n",i,vx[i]); /*エラー処理*/
      }
   }
   return(0);
}/*main関数の終わり*/
```

Level	Comment	Generated Procedure	Process
1	配列の添え字を定義 Define number of array	配列の添え字，最小値，最大値の定義 Definition of array subscript, minimum value, maximum value	並列の統合 Parallel integration
1	最小値の定義 Definition of minimum value		
1	最大値の定義 Definition of maximum val ue		
1	main関数の宣言 Declaration of main function		削除
2	整数型変数の宣言 Declaration of integer variable	整数型配列変数の宣言 Declaration of integer type array variable	並列の統合 Parallel integration
2	整数型変数の配列を宣言 Declare array of integer variable		
2	数値の入力 Entering Numbers	配列に数値の入力 Enter numerical value in array	F値の優先 Selection by F v alue
3	配列に数値を入力 Enter numeric value in array		
2	異常値の出力 Output of outliers		F値による削除 Delete by F value
2	配列内の確認 Confirm in the array	配列内の確認 Confirm in the array	F値の優先 Selection by F value
3	最小値から最大値でない場合 Case that is not the minimum value to the maximum value	最小値から最大値でない場合 Case that is not the minimum value to the maximum value	
4	エラー処理 Error handling	エラー処理 Error handling	
1	main関数の終わり End of main function		削除 Delete

Figure 4: Extraction of important comments by F-value



Program code(C programing language)

```
/*配列内のエラーの個数*/
#include <stdio.h>
#define NUMBER 5 /*配列の添え字を定義*/
#define MIN 0 /*最小値の定義*/
#define MAX 100   /*最大値の定義*/

int main(void){   /*main関数の宣言*/
   int i,j; /*整数型変数の宣言*/
   int vx[NUMBER]; /*整数型変数の配列を宣言*/

   /*数値の入力*/
   for(i=0;i<NUMBER;i++){
      printf("vx[%d]=",i); /*配列に数値を入力*/
      scanf("%d",&vx[i]);
   }
   /*異常値の出力*/
   printf("エラー値の出力¥n");
   for(i=0; i<NUMBER; i++){   /*配列内の確認*/
      if(!(vx[i] >= MIN && vx[i] <=MAX)){ /*最小値から最大値でない場合*/
         printf("vx[%d]=%dはエラーです¥n",i,vx[i]); /*エラー処理*/
      }
   }
   return(0);
}/*main関数の終わり*/
```

Generated Comments

(1) Define the size, minimum value, maximum value of the array.
  配列の大きさ、最小値、最大値を定義する
(2) Declare the Main function.
  Main関数を宣言する
(3) Declare an integer type array.
  整数型の配列を宣言する。
(4) Enter numerical values in all subscripts of the array.
  配列のすべての添え字内に数値を入力する
(5) Case that is not within the range from the minimum value to the maximum value
  最小値から最大値内でない場合
(6) Error handling
  エラー処理

Figure 5: Comment synthesis results

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 1: Evaluation of generated procedure by F-value

Questionnaire	Evaluation(1-5)				
	1	2	3	4	5
(1) Evaluation of procedure	0%	27%	62%	11%	0%
(2) Program understanding related to the procedure	0%	46%	43%	9%	2%

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, sentnce 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.

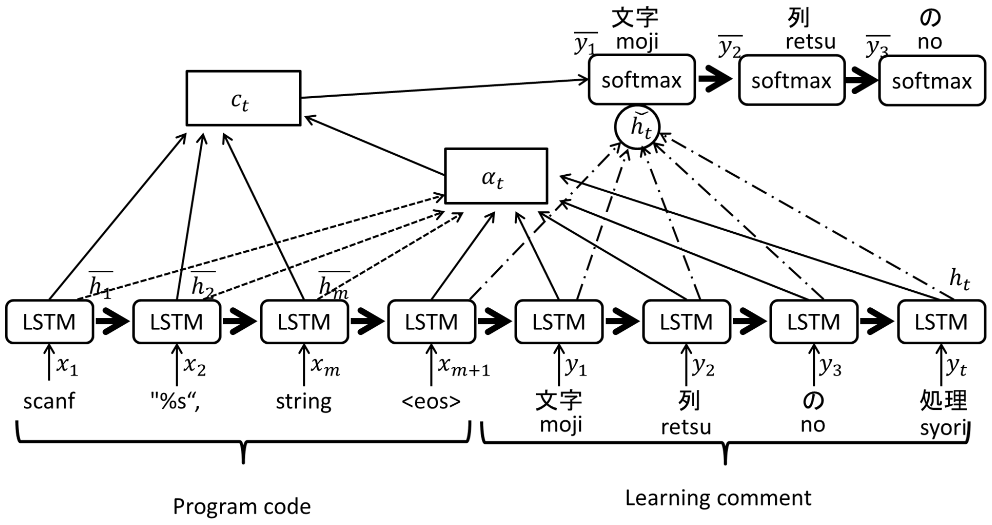


Figure 6: Encoder–decoder translation model for neural machine translation from comments to procedure

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 α_t, a normalization of the degree of similarity is performed for output ht in relation to y_t, and $\overline{h_i}$ on the encoder side. c_t creates context vectors using the degree of similarity obtained with α_t and $\overline{h_t}$.
(2) In the LSTM output using the encoder–decoder translation model, c_t and $\overline{h_t}$ 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 $\check{h}_i$ is output in relation to y_t. Finally, a weight is applied with a linear operator with respect to $\check{h}_i$, and $\overline{y}_t$ is output with the softmax function.

4.3 Data learning with a source code comments pair

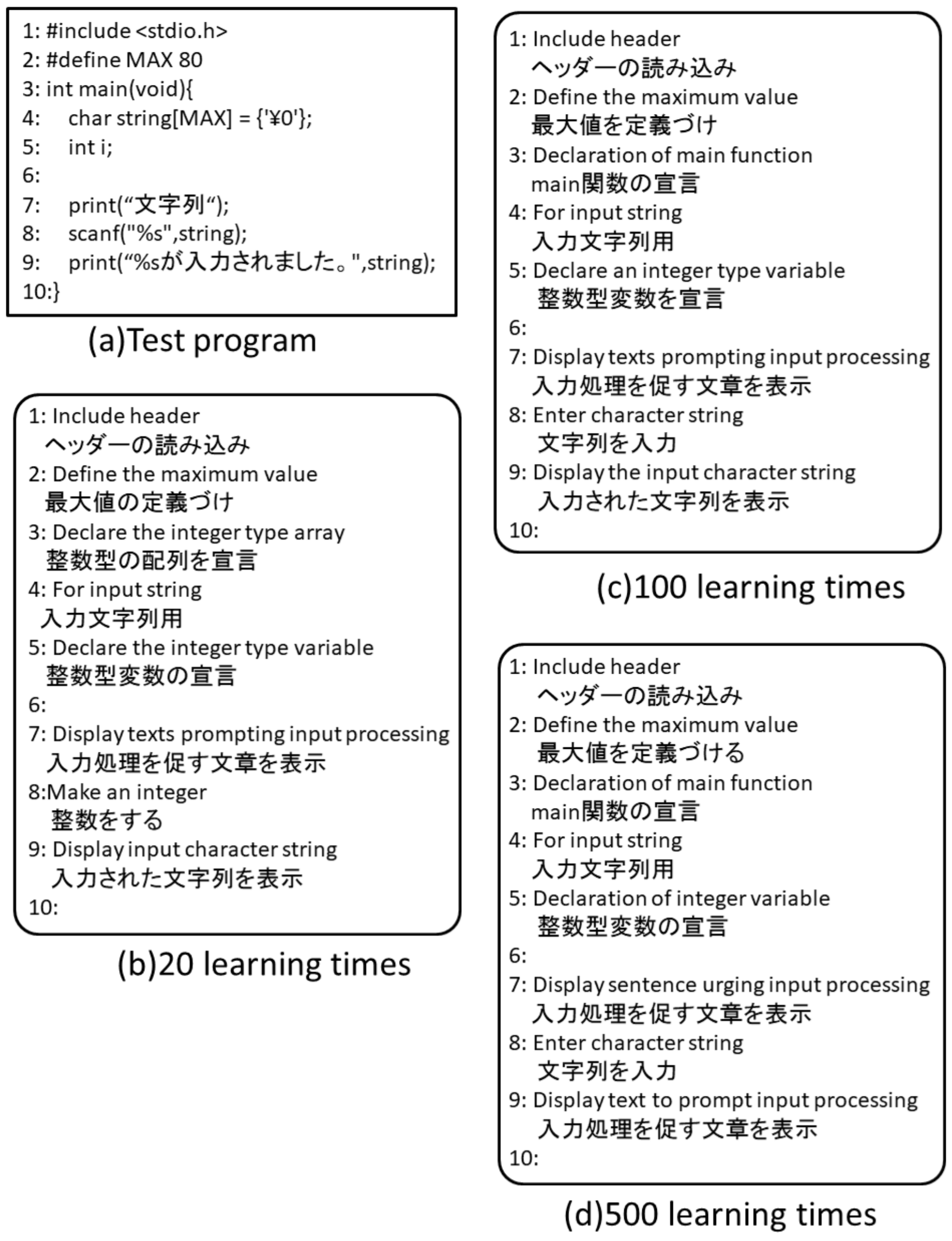
Learning is performed multiple times using the defined procedure generation model. LSTM is used for learning data, and screening occurs using the sigmoid layer to select whether input x_i is required. Furthermore, the loss with the accuracy data that was saved in advance is obtained. The accumulated 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.



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

(a)Test program

```
1: Include header
  ヘッダーの読み込み
2: Define the maximum value
  最大値の定義づけ
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 the integer type variable
  整数型変数の宣言
6:
7: Display texts prompting input processing
  入力処理を促す文章を表示
8:Make an integer
  整数をする
9: Display input character string
  入力された文字列を表示
10:
```

(b)20 learning times

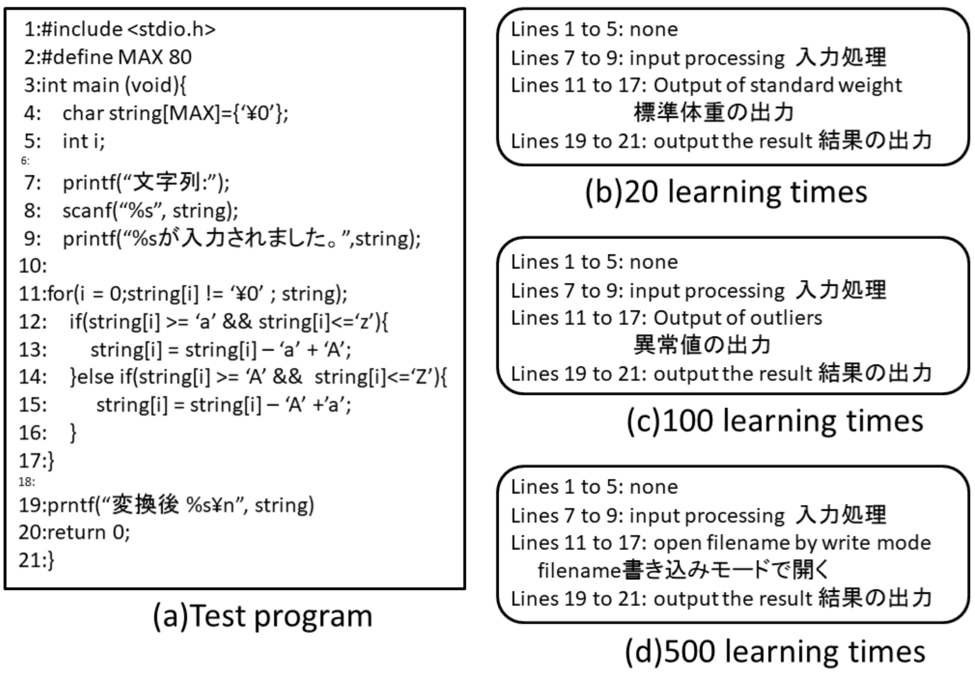
```
1: Include header
  ヘッダーの読み込み
2: Define the maximum value
  最大値を定義づけ
3: Declaration of main function
  main関数の宣言
4: For input string
  入力文字列用
5: Declare an integer type variable
  整数型変数を宣言
6:
7: Display texts prompting input processing
  入力処理を促す文章を表示
8: Enter character string
  文字列を入力
9: Display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

(c)100 learning times

```
1: Include header
  ヘッダーの読み込み
2: Define the maximum value
  最大値を定義づける
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:
```

(d)500 learning times

Figure 7: Example of procedure generations


(a)Test program
(b)20 learning times
(c)100 learning times
(d)500 learning times

```
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:
11:for(i = 0;string[i] != '¥0' ; string);
12:    if(string[i] >= 'a' && string[i]<='z'){
13:        string[i] = string[i] – 'a' + 'A';
14:    }else if(string[i] >= 'A' &&  string[i]<='Z'){
15:        string[i] = string[i] – 'A' +'a';
16:    }
17:}
18:
19:prntf("変換後 %s¥n", string)
20:return 0;
21:}
```

(b)
Lines 1 to 5: none
Lines 7 to 9: input processing 入力処理
Lines 11 to 17: Output of standard weight 標準体重の出力
Lines 19 to 21: output the result 結果の出力

(c)
Lines 1 to 5: none
Lines 7 to 9: input processing 入力処理
Lines 11 to 17: Output of outliers 異常値の出力
Lines 19 to 21: output the result 結果の出力

(d)
Lines 1 to 5: none
Lines 7 to 9: input processing 入力処理
Lines 11 to 17: open filename by write mode filename書き込みモードで開く
Lines 19 to 21: output the result 結果の出力

Figure 8: Procedure generation for each block of source code by encoder–decoder translation model

Table 2: Evaluation of generated procedure by line

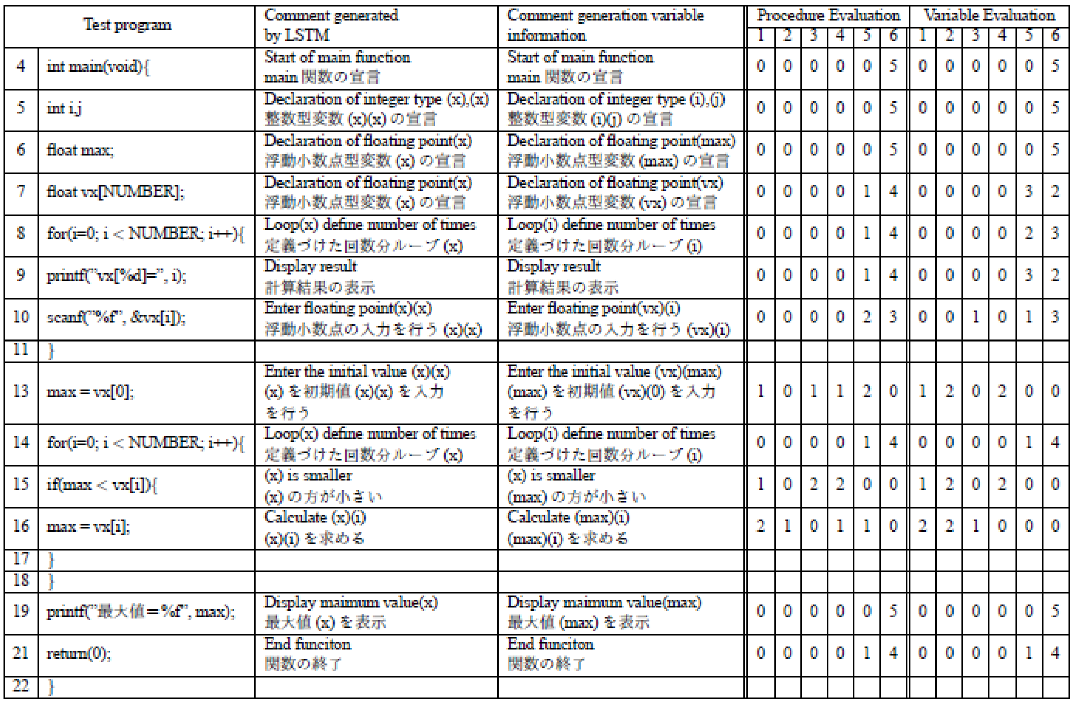
Line number	Generated procedure	Evaluation(1-6)					
		1	2	3	4	5	6
1	Include standard input output file 標準入力ファイルの読込	0	0	0	0	0	5
2	Definition of maximum value 最大値の定義	0	0	0	0	0	5
3	Declaration of main main の宣言	0	0	0	0	0	5
4	For input string 入力文字列用	0	0	3	2	0	1
6	Declare the integer type variable 整数型変数を宣言	0	0	0	0	0	5
7	Display text prompting input processing 入力処理を促す文章を表示	0	0	0	0	0	5
8	Enter a character string 文字列を入力	0	0	0	0	0	5
9	Display the input character string 入力された文字列を表示	0	0	0	0	0	5

Table 3: Evaluation of generated procedure by each block

Line number	Generated procedure	Evaluation(1-6)					
		1	2	3	4	5	6
1–5	none	1	0	0	1	0	3
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 $\rightarrow$ x) and i for x(i $\rightarrow$ 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.



Test program		Comment generated by LSTM	Comment generation variable information	Procedure Evaluation						Variable Evaluation					
				1	2	3	4	5	6	1	2	3	4	5	6
4	int main(void){	Start of main function main 関数の宣言	Start of main function main 関数の宣言	0	0	0	0	0	5	0	0	0	0	0	5
5	int i,j	Declaration of integer type (x),(x) 整数型変数 (x)(x) の宣言	Declaration of integer type (i),(j) 整数型変数 (i)(j) の宣言	0	0	0	0	0	5	0	0	0	0	0	5
6	float max;	Declaration of floating point(x) 浮動小数点型変数 (x) の宣言	Declaration of floating point(max) 浮動小数点型変数 (max) の宣言	0	0	0	0	0	5	0	0	0	0	0	5
7	float vx[NUMBER];	Declaration of floating point(x) 浮動小数点型変数 (x) の宣言	Declaration of floating point(vx) 浮動小数点型変数 (vx) の宣言	0	0	0	0	1	4	0	0	0	0	3	2
8	for(i=0; i < NUMBER; i++){	Loop(x) define number of times 定義づけた回数分ループ (x)	Loop(i) define number of times 定義づけた回数分ループ (i)	0	0	0	0	1	4	0	0	0	0	2	3
9	printf("vx[%d]=", i);	Display result 計算結果の表示	Display result 計算結果の表示	0	0	0	0	1	4	0	0	0	0	3	2
10	scanf("%f", &vx[i]);	Enter floating point(x)(x) 浮動小数点の入力を行う (x)(x)	Enter floating point(vx)(i) 浮動小数点の入力を行う (vx)(i)	0	0	0	0	2	3	0	0	1	0	1	3
11	}														
13	max = vx[0];	Enter the initial value (x)(x) (x) を初期値 (x)(x) を入力を行う	Enter the initial value (vx)(max) (max) を初期値 (vx)(0) を入力を行う	1	0	1	1	2	0	1	2	0	2	0	0
14	for(i=0; i < NUMBER; i++){	Loop(x) define number of times 定義づけた回数分ループ (x)	Loop(i) define number of times 定義づけた回数分ループ (i)	0	0	0	0	1	4	0	0	0	0	1	4
15	if(max < vx[i]){	(x) is smaller (x) の方が小さい	(x) is smaller (max) の方が小さい	1	0	2	2	0	0	1	2	0	2	0	0
16	max = vx[i];	Calculate (x)(i) (x)(i) を求める	Calculate (max)(i) (max)(i) を求める	2	1	0	1	1	0	2	2	1	0	0	0
17	}														
18	}														
19	printf("最大値=%f", max);	Display maimum value(x) 最大値 (x) を表示	Display maimum value(max) 最大値 (max) を表示	0	0	0	0	0	5	0	0	0	0	0	5
21	return(0);	End funciton 関数の終了	End funciton 関数の終了	0	0	0	0	1	4	0	0	0	0	1	4
22	}														

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

[1] Ministry of Education, Culture, Sports, Science and Technology, "Elementary school programming education guide (2nd edition)," 2018. [Online] Available: http://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm, [Accessed Nov. 15, 2018] (in Japanse).

[2] Ministry of Education, Culture, Sports, Science and Technology. "How to programming education at elementary school level (Summary of discussion)." 2016. [Online] Available: http://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm , [Accessed Nov. 15, 2018] (In Japanese).

[3] H. Kanamori, T. Tomoto and T. Akakura, "Development of a Computer Programming Learning Support System Based on Reading Computer Program," *Human Interface and the Management of Information. Information and Interaction for Learning, Culture, Collaboration and Business (HIMI) 2013*, pp. 63-69, Springer, 2013. DOI:10.1007/978-3-642-39226-9_8

[4] K. Okimoto, S. Matsumoto, S. Yamagishi and T. Kashima, "Developing a source code reading tutorial system and analyzing its learning log data with multiple classification analysis," *Artificial Life and Robotics*, Vol 22, No. 7, pp. 227-237, 2017. DOI:10.1007/s10015-017-0357-2

[5] S. Matsumoto, K. Okimoto, T. Kashima and S. Yamagishi, "Automatic Generation of C Source Code for Novice Programming Education, Human-Computer Interaction," *Theory, Design, Development and Practice 2016*, pp. 65-76, 2016. DOI:10.1007/978-3-319-39510-4_7

[6] M. Oba, K. Ito, and A. Shimogoori, "Analysis of Correlation between Programming Skills and Technical Writing Skills," *IPSJ SIG Technical Report*, Vol 2015-IFAT-118 No. 2, pp. 1-4, 2015.

[7] K. Sakane, N. Kobayashi, H. Shiina and F. Kitagawa, "Kanji Learning and Programming Support System which conjoined with a Lecture," *IEICE Technical Report*, ET2014-86, Vol. 114, No. 513, pp. 7-12, 2015.

[8] F. Tetsuya, Y. Hayase and K. Inoue, "Generating Descriptions of Nouns in Software from Program Comments," IEICE-110, no. 169, pp. 65-69, 2010.

[9] I.Mani and E. Bloedorn, "Multi-document summarizzation by graph search and matching," In Proc. 14th National Conferenece on Artificial Intelligence, pp. 622-628 1997.

[10] D. Marcu, "Improving summarization through rhetorical parsing tuning," In Proc. 6th Workshop on Very Large Corpora, pp.206-215, 1998.

[11] I. Mani. *Automatic Summarization*, John Benjamins Pub Co, 2001. DOI:10.1075/nlp.3

[12] K. Greff, et al., "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 28, Issue10, pp. 2222-2232, 2017.

[13] I. Sutskever, O. Vinyals and Q. Le, “Sequence to Sequence Learning with Neural Networks,” Advances in Neural Information Processing Systems 27 (NIPS 2014), pp. 3104-3112, 2014.

[14] M. Luong, H. Pham and D. Manning, “Effective Approaches to Attention-based Neural Machine Translation,” arXiv preprint arXiv:1508.04025v5, 2015.

[15] A. Rush, S. Chopra and C. Weston, “A Neural Attention Model for Sentence Summarization,” In Proc. EMNLP 2015: Conference on Empirical Methods in Natural Language Processing, pp. 379-389, 2015.