Proc. (Vo.1) of International Conference on Computers in Education 99 (ICCE '99), pp.776-783 (1999).

Making Fill-in-Blank Program Problems for Learning Algorithm

Akihiro Kashihara, Atsuhiro Terai, and Jun'ichi Toyoda

Making Fill-in-Blank Program Problems for Learning Algorithm

Akihiro Kashihara, Atsuhiro Terai, and Jun'ichi Toyoda

I.S.I.R., Osaka University, JAPAN
8-1, Mihogaoka, Ibaraki, Osaka 567-0047, JAPAN
Tel. +81-6-6879-8426 kasihara@ai.sanken.osaka-u.ac.jp

Fill-in-blank program problem gives learners a program of which part is blanked out and the program specification. They are required to fill in the blank so that the program specification can be fulfilled. In solving the problem, they need to trace data and control flow of the program. This induces them to think of algorithm embedded in the program, enhancing their learning. However, whether learning is effectively enhanced depends on how to make a blank. This paper proposes a method of blanking out an important point of data or control flow of a program to make instructive fill-in-blank problem. The essence of this method is to find out the important point with Program Dependence Graph in no consideration of semantic aspects of the algorithm. It can be consequently incorporated into computer-based learning systems. This paper also describes a preliminary experiment on the blank-making method with subjects who are familiar with algorithms. In this experiment, we have ascertained that blanks made by hand follow the blank-making method.

Keywords: IT Education, Teaching/Learning Strategies, Intelligent Tutoring

1. Introduction

How to assist learners in learning algorithms is an important topic in computer science education. Current work on computer-based learning support has provided a number of aids such as algorithm animation [1, 2]. We have also followed this topic with another approach that uses fill-in-blank program problem[4, 8].

The fill-in-blank program problem gives learners a program of which part is blanked out and the program specification. They are required to fill in the blank so that the program specification can be fulfilled. In solving the problem, they would trace data and control flow of the program. This induces them to make cognitive efforts at thinking of algorithm that specifies the process flow. These cognitive efforts would enhance their learning of the algorithm [5]. In fact, we have ascertained experimentally that fill-in-blank problems improve comprehension of algorithm [4]. However, learners may have an overload on problem solving. It is therefore necessary to blank out part of program in a proper way.

The main issue addressed in this paper is how to make a blank in a program. The important point toward instructive fill-in-blank problem is to allow learners to have more chances to think of the algorithm in problem solving. This requires blanking out an important point of data or control flow of the program.

Using a simple algorithm with a program written by C programming language, this paper describes a method of finding out an important point of process flow of the program with Program Dependence Graph (PDG for short) [3]. The essence of this method is to make a blank in no consideration of semantic aspects of the algorithm. Therefore, it would be easy to incorporate the blank-making method into computer-based learning systems. Intended learners have an insufficient understanding of algorithms, and have knowledge of the syntax of C programs.

[Problem] The following program sorts the list tbl[i] (i=1...n) in ascendant order.
Fill in the blank to complete the program.

```
#include <stdio.h>
#define MAX 5
main ()
{
  int i, j;
  int tbl[MAX], n;
  for (i=0;i<MAX; i++) {
    printf("Input number tbl[%d] = ", i);
    scanf("%d", &tbl[i]); }
  for (i=0; i<MAX; i++){
    for(j=i+1; j<MAX; j++)
      if (              ){
        n = tbl[j];
        tbl[j] = tbl[i];
        tbl[i] = n; } }
  for (i=0; i<MAX; i++)
    printf("tbl[%d] = %d\n", i, tbl[i]);
}
```

Figure 1: A Fill-in-Blank Program Problem.

This paper also describes a preliminary experiment with subjects who are familiar with algorithms. In this experiment, we have ascertained whether blanks made by hand follow the blank-making method. As a result, the validity of the method has been demonstrated overall.

2. Fill-in-Blank Program Problem

Before discussing how to make a blank in a program, let us give an overview of fill-in-blank program problem.

Figure 1 shows an example of fill-in-blank problem. The role of the blank is to break the data or control flow of the program. Filling in the blank accordingly corresponds to complementing the broken flow, which involves looking around uncovered program codes to trace the process flow. This induces learners to think of algorithm embedded in the program. The fill-in-blank problem therefore enables them to promote their understanding of the algorithm.

Let us next explain the effectiveness of fill-in-blank problem as compared with existing representative aids such as algorithm animation and flowchart. Algorithm animation demonstrates data and control flow of algorithm in a dynamic manner [2]. Flowchart also shows learners the process flow in a static manner [7]. These aids enable learners to acquire knowledge of the algorithm with less difficulty [6]. On the other hand, the fill-in-blank problem encourages learners to make cognitive efforts to trace and identify the algorithm from the program [8]. These cognitive efforts would deepen and enhance their learning of the algorithm [5].

Although we can also consider a flowchart with blanks, it is less difficult since the flowchart represents the skeleton of the process flow of algorithm. The fill-in-blank program problem is accordingly more proper for the learning enhancement since it requires learners to find out the process flow with the skeleton from the program.

However, whether learning is effectively enhanced depends on where to blank out. Let us now discuss the blank-making method in the following.

3. How to Make a Blank

3.1 Restriction

Let us first describe the restriction on blank making. In the fill-in-blank problem, the size and number of blanks in addition to the position have a great influence on problem solving. We accordingly restrict them to make fill-in-blank problem with moderate difficulty.

3.1.1 Size

We can make a blank that includes one variable, one statement, or one block in a program. The larger the size of the blank, the more difficult tracing the process flow of the program. It is also difficult for learners to focus on understanding the algorithm since they have to make more efforts to program in the blank. The larger size of the blank is consequently unsuitable for helping learners who have an insufficient understanding of algorithm. We accordingly limit the size of the blank to one statement.

3.1.2 Number of Blanks

In case there are a number of blanks in a program even if the size of each blank is small, tracing the process flow of the program is more difficult since it is broken here and there. Considering that intended algorithms are all simple, we accordingly limit the number of blanks to two.

In our previous work [4], we proposed the method of making two blanks called main and sub blanks. The main blank was made at a statement corresponding to an important position of algorithm, which was decided by hand in consideration of semantic aspects of the algorithm. The sub blank was made at the statement to which the main blank was relevant. There were some kinds of relevance between the blanks: adjacency, within the same block, etc. The sub blank was formally placed once the position of the main blank was decided. The objective of making the sub blank was to change the difficulty in filling in the main blank.

On the other hand, this paper follows the above restriction to propose a method of making one blank in no consideration of semantic aspects of algorithm, which blank corresponds to the main blank in the previous work.

3.2 PDG

The considerable issue toward instructive fill-in-blank problem is where to blank out so that learners can have more chances to think of process flow of the program. A promising approach to this issue is to blank out part of program that is an important point of the process flow. From this point of view, let us discuss how to find out the important point with PDG.

PDG is a directed graph of which nodes represent statements in a program and arcs represent dependence between statements. There are two kinds of dependence: (i) data dependence and (ii) control dependence.

Data dependence is defined as follows.

> When two statements *s1* and *s2* have the following relationships, there exists data dependence about variable *v* from *s1* to *s2*.
> - Variable *v* is defined in *s1*.
> - Variable *v* is not redefined in the feasible paths between *s1* and *s2*.
> - Variable *v* is referred to in *s2*.

In Figure 2 (a), for example, data dependence about variable x exists from line 2 to line 3. However, the data dependence does not exist from line 1 to line 3 since variable x is redefined between these lines.

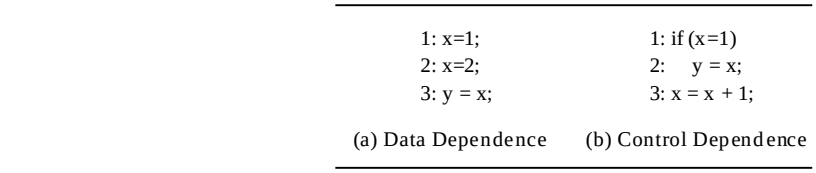

Figure 2: Data Dependence and Control Dependence.

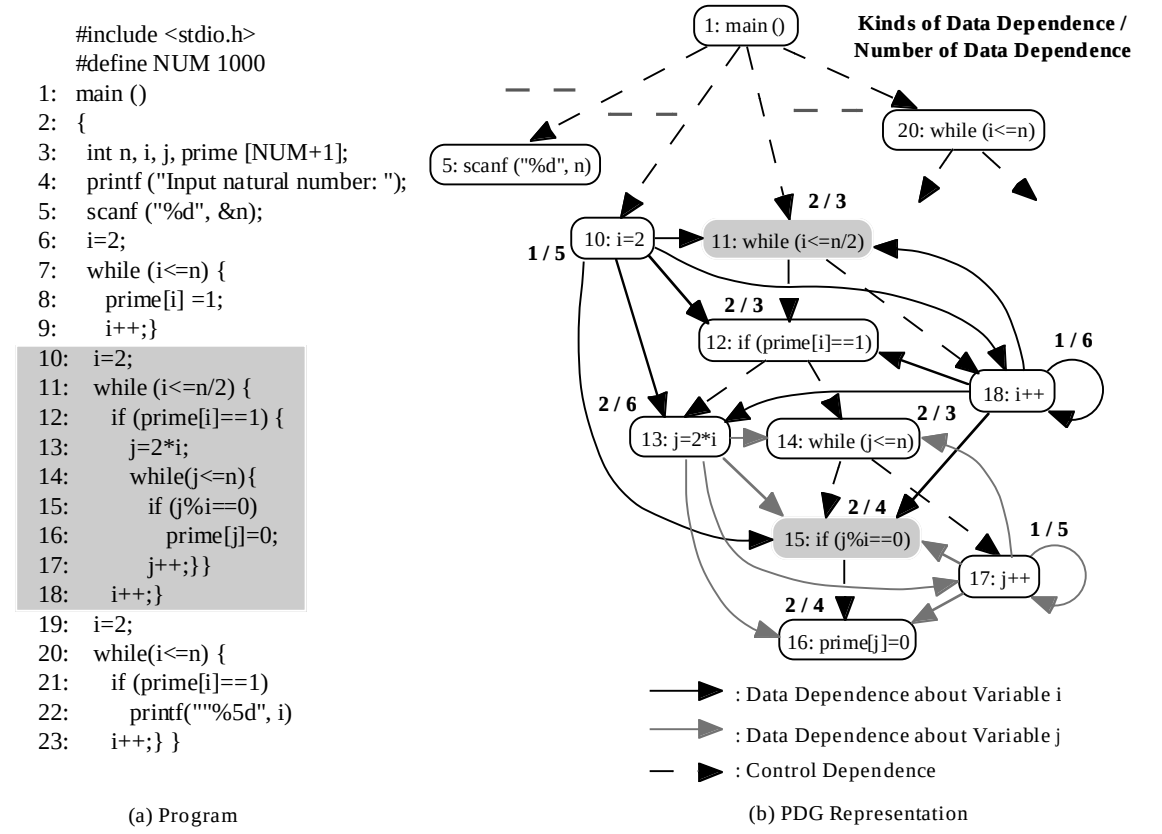

Figure 3: A Program for Finding Out Prime Numbers.

Control dependence is also defined as follows.

> When two statements *s1* and *s2* have the following conditions, there exists control dependence from *s1* to *s2*.
> - *s1* is a conditional or while-loop statement.
> - Whether *s2* is executed or not depends on the result of execution of *s1*.

In Figure 2 (b), for example, control dependence exists from line 1 to line 2.

Figure 3 shows a program, which embeds the algorithm of finding out prime numbers, and the PDG that represents the program statements from line 10 to line 18. As shown in this figure, data dependence in the program exists each variable. Figure 3 (b) shows data dependence only about variables i and j. Control dependence in the program often forms a nested structure called control structure in which controlled statements recurrently become control statements.

3.3 Where to Make a Blank

In PDG, data flow of a program can be quantified with the number and kinds of data dependence arcs going in and out nodes. A node with more arcs can be viewed as a more

important statement in the data flow. Blanking out the statement would cause learners to refer to more statements, which have data dependence with the blank, in solving the problem. Control statements such as if-clause and while-clause can be also viewed as important statement in the process flow since these manage not only control flow but also data flow of the program. In particular, blanking out a control statement in a lower position in the control structure would cause learners to think of the details of the data and control flow.

From the above point of view, we regard node that fulfills the following conditions with logical products as important position of process flow of program.

- Node, which represents a control statement, becomes important position when
  - it is in lower position in control structure, AND
  - it has more kinds of data dependence arcs, AND
  - it has more data dependence arcs.
- Node, which represents a non-control statement, becomes important position when
  - it has more kinds of data dependence arcs, AND
  - it has more data dependence arcs.

Let us consider important positions in Figure 3 (b). The kinds and number of data dependence arcs going in and out are attached to each node. A candidate for important position as control statement is the node of line 15 since it is in the lowest position in the control structure and it has the most kinds and number of data dependence arcs in the control structure. As for important position as non-control statement, the node of line 13 is regarded as the first candidate. The node of line 16 is also regarded as the second candidate. The algorithm embedded eliminates multiples of each number from 2 to n/2 to regard the remaining numbers as prime numbers. The important aspects of this algorithm from a semantic point of view are:

(A) to check whether the multiples are in the range of numbers to n.
(B) to decide the rage in which the multiples should be eliminated.
(C) to decide the rage of numbers to be multiplied.
(D) to set or reset the flag that indicates whether numbers are prime ones.

Except for (C), these correspond to the important positions found out with PDG. The node of line 15 corresponds to (A); the node of line 13, (B); and the node of line 16, (D). This suggests that the blank making method with PDG is valid.

4. Preliminary Evaluation

4.1 Experiment

In order to evaluate the validity of the blank making method, we had an experiment. In this experiment, we acertained if blanks made by hand followed the blank making method. Subjects were 10 university teachers, graduate and undergraduate students in science and technology who were familiar with programming and algorithms.

We prepared three algorithms and their programs, which were: algorithm of finding out prime numbers, algorithm of sorting numbers, and algorithm of calculating common multiples. In the experiment, each subject was required to confirm each algorithm with flowchart. He/she was then asked which statement in each program should be blanked out so that a novice learner can promote learning of the algorithm. He/she selected a statement as blank twice (After he/she selected a statement, he/she was again asked to select another statement from the original program.)

Table 1: Results of Experiment.

Algorithm	Selected Statement	Selected Times	Control/ Non-Control Statement	Position in Control Structure	Largeness of Kinds/Number of Data Dependence
Prime Numbers	**First:** if (j%i==0) [Line15]	7	Control	Deepest	Largest / Largest
	Second: while (i<=n/2) [Line11]	4	Control	Unrelated	Largest / Second
Sorting	**First:** while(j>=1&&b[j]<b[j-1]) [Line12]	9	Control	Deepest	Largest / Largest
	Second: j=i [Line11]	3	Non-Control	———	Second / Largest
Common Multiples	**First:** while(i*m<MAX) [Line16]	6	Control	Unrelated	Second / Second
	Second: if(i*m%p[j]==0) [Line20]	5	Control	Deepest	Largest / Largest

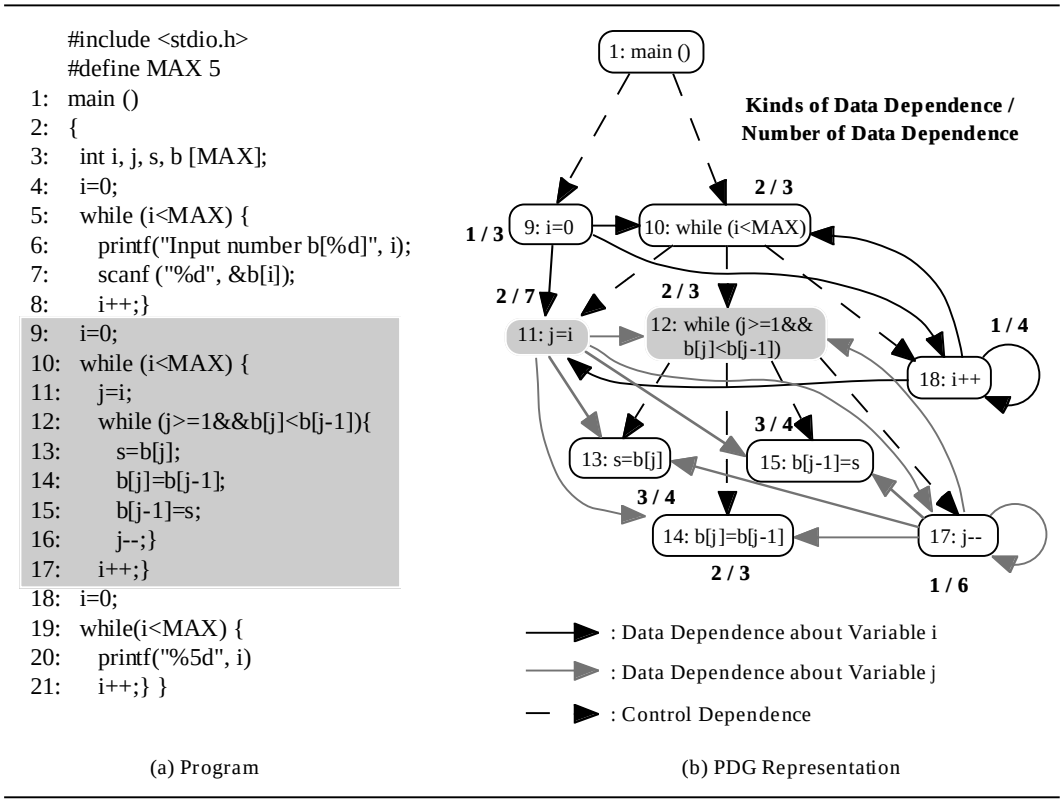

Figure 4: A Program for Sorting Numbers in Ascendant Order.

4.2 Results and Discussion

Table 1 shows the results of the experiment. In this table, we present the statements of which selected times were ranked in the first and second positions among statements selected as blank. These statements are represented as mesh nodes in Figure 3, Figure 4, and Figure 5, which figures show the programs used in the experiment and the PDGs. The table also describes whether selected statement is a control statement or not, the position in control structure, and the largeness of kinds and number of data dependence arcs.

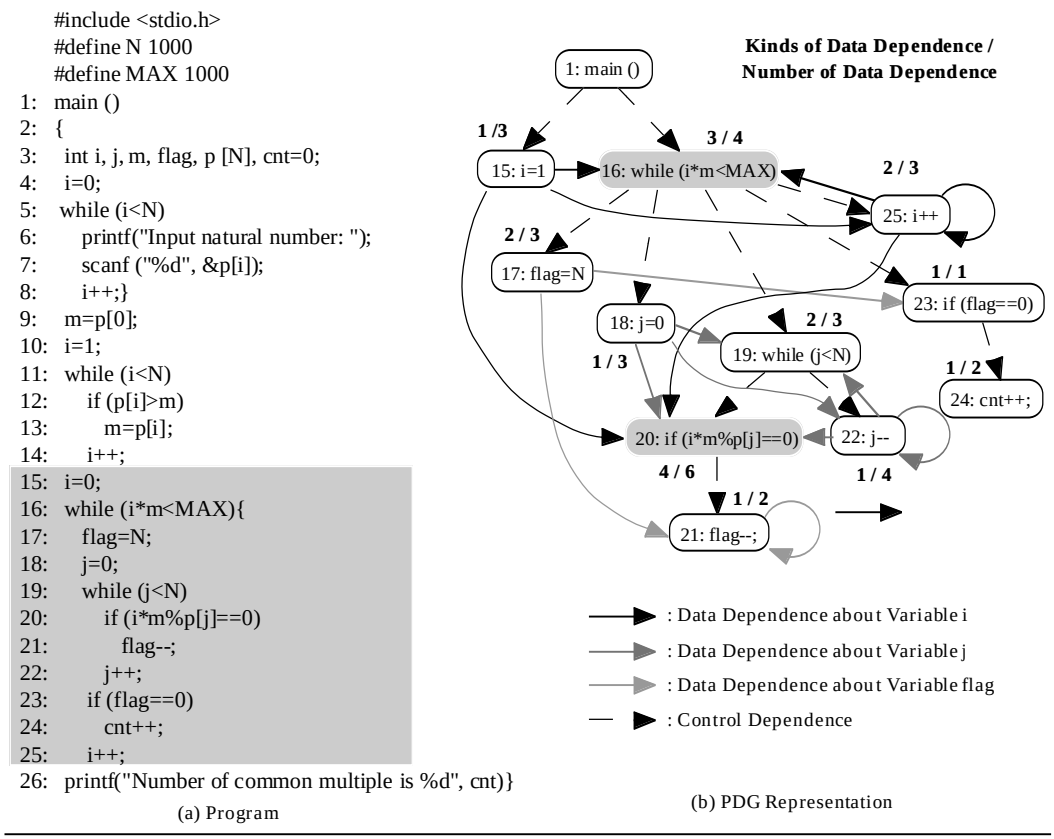

Figure 5: A Program for Calculating Common Multiples.

The results suggest that the blank-making method is valid overall. As shown in Table 1, the node of which number and kinds of data dependence arcs is the largest or second largest of all statements in the program tends to be selected as blank. As expected, positions to be regarded as important from a PDG point of view are blanked out. The control statements also tend to be blanked out. We can consider the reason is that blanking out control statement induces learners to pay more attention to the process flow of the program.

However, the results do not make it clear whether position in control structure could be a condition for making a blank since there exist two statements in this table which are not in a lower position in the control structure. We need to conduct more detailed experiment to ascertain it.

5. Conclusion

This paper has described a method of making instructive fill-in-blank program problem for learning algorithm embedded in the program. This method finds out an important point of process flow of the program with PDG, and blanks it out. The essence of this method is to make a blank in no consideration of semantic aspects of the algorithm. Therefore, this blank-making method can be incorporated into computer-based learning systems, which is a main future work.

This paper has also described a preliminary experiment on the blank-making method. The results have suggested that it is valid overall. In the future, we would have more detailed evaluation to refine the proposed method.

References

[1] Brown, M.H.: Zeus: A System for Algorithm Animation and Multi-View Editing, Proc. of the 1991 IEEE Workshop on Visual Languages, pp.4-9 (1991).
[2] Hansen, S.R., Narayanan, N.H., and Schrimpsher, D.: Rethinking Algorithm Animation: A Framework for Effective Visualization, Proc. of ED-MEDIA 98, pp. 1653-1654 (1998).
[3] Horwitz, S., Reps, T., and Binkley, D.: Interprocedural Slicing Using Dependence Graphs, ACM Trans. on Programming languages and Systems, Vol.12, No.1, pp.26-60 (1990).
[4] Kashihara, A., Soga, M., and Toyoda, J.: A Support for Program Understanding with Fill-in-Blank Problems, Transactions of Japanese Society for Information and Systems in Education, Vol. 15, No.3, pp. 129-138 (1998 in Japanese).
[5] Kashihara,A., Hirashima,T., & Toyoda,J.: A Cognitive Load Application in Tutoring. Journal of User Modeling and User-Adapted Interaction, Vol.4, No.4, pp.279-303 (1995).
[6] Merrill,D.C., Reiser,B.J., Beekelaar,R., & Hamid.A., (1992). Making Processing Visible: Scaffolding Learning with Reasoning-congruent Representations. Proc. of 2nd International Conference on ITS (ITS '92), pp. 103-110 (1992).
[7] Scanlan, D.A.: Structured Flowcharts Outperform Pseudocode: An Experimental Comparison, IEEE Software, Vol.6, No.5, pp.28-36 (1989).
[8] Soga, M., Kashihara, A., and Toyoda, J.: Adaptive Fill-in-blank Program Problems from The View of Cognitive Load and Application Systems on WWW, Proc. of ICCE'95, pp.575-582 (1995).