International Journal of Advanced Research in Big Data Management System
Vol. 2, No. 1 (2018), pp.7-12
http://dx.doi.org/10.21742/ijarbms.2018.2.1.02

Development of Problem Solving Path Algorithm for Individualized Programming Education

Youngho Lee

Seoul Metropolitan Office of Education
yhlee1627@gmail.com

Abstract

Computational thinking is important all over the world. Students are taught programming to improve their computational thinking. Both block-type and text-type programming languages are introducing tutorial learning methods. In order to increase the educational effect of this method, the learner should be able to solve each mission of the tutorial by himself. To do this, the system needs to provide feedback appropriate to the level so that the learner can solve the problem. However, the current tutorial system is a static feedback method that provides feedback based on the number of blocks and the presence or absence of blocks compared to the correct blocks. In this study, the algorithm is designed that extracts problem solving paths for each level by using the students' problem solving paths and verifies its performance. This study suggests a new method for customized learning using existing learner data in programming education.

Keywords: *Programming education, individualized learning, graded learning trajectory, learning data.*

1. Introduction

Various educational programs are conducted with the aim of helping learners develop computational thinking abilities particularly through programming[8]. Various methods to educate learners about such programming have been examined and applied to develop block-type or text-type (textual) programming languages for educational purposes[6]. Such learning mechanisms introduce tutorial learning methods in both text-type and block-type programming languages[8]. In order to enhance the educational effects of such tutorial methods, learners need to be induced to face each tutorial in a self-directed manner. Platforms that adopt tutorial methods provide learners with a hint when necessary so that they can complete each mission[2][7]. In the process, learners are given feedback regarding the number of blocks, [1]incorrect answers, et cetera, but a process to analyze learners is not involved. Specifically, the thinking process that each learner goes through to provide blocks is not taken into consideration. There is a limitation in providing each learner with proper feedback since static learning feedback is given instead of dynamic learning feedback. This study aims to develop an algorithm to predict a problem-solving trajectory according to each learner's level and give customized feedback. Specifically, a method to provide learners with feedback in a social constructivism perspective is proposed. Vygotsky (1980) suggested the theory of a proximal development zone that indicates the distance between a substantial development

Article history:
Received (February 11, 2018), Review Result (February 26, 2018), Accepted (April 21, 2018)

Print ISSN: 2208-1674, eISSN: 2208-1682 IJARBMS

level of independent problem-solving and a potential development level of solving problems with help from teachers or in cooperation with more capable companions. Accordingly, this study seeks to develop the algorithm predict customized learning path based on learners' problem-solving trajectory data.

2. Data overview

At the website of "The Hour of Code," users can program maze puzzle missions[2]. The data used in this study is that of Hoc (Hour of Code) Level 4 and Level 18 as shown in Figure 1.

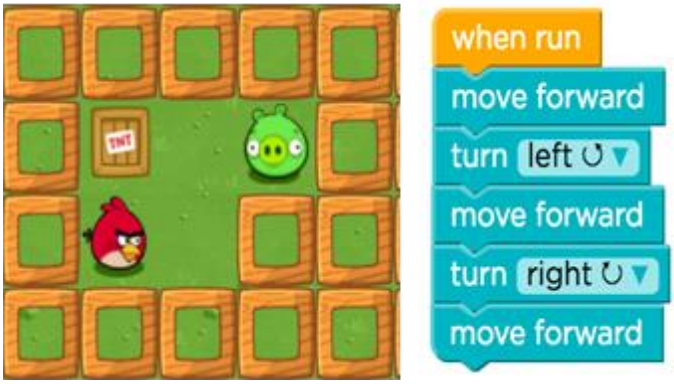

Figure 1. Hoc 4 Questions and Answers at Code.org

Figure 2. Hoc 18 Questions and Answers at Code.org

Hoc 4 is a question regarding the concept of sequence where a problem is solved using forward blocks and left/rightward revolving blocks. Hoc 18 is a question regarding the concepts of loop and condition (if/else). The level of difficulty is the highest among the missions of "The Hour of Code.". Data Submitted by Every Learner Logged onto the Hour of Code between December 2013 and March 2014. There is difference in the quantity of submitted codes depending on the level of problem difficulty. While the number of blocks used in problem-solving was six or less, the number of responses submitted by learners was 10,293 in the case of Hoc 4 and 79,553 in the case of Hoc 18. The number of problem-solving trajectories used in this study as important data was 55,351 in the case of Hoc 4 and 83,954 in the case of Hoc 18. In the case of Hoc 4, which

presented questions of a relatively easy concept, the success rate of learners was 97.8%, while in the case of Hoc 18, which presented questions of a relatively complicated concept, the success rate was 81.0%[1].

Table 1. Data sets used in the research

Statistics	Hoc 4	Hoc 18
Unique Students	509,405	263,569
Code Submissions	1,138,506	1,263,360
Trajectories	55,351	83,954
Pass Rate	97.8%	81.0%

3. The algorithm to generate problem-solving graphs

Problem-solving graph is generated as graphs based on the data sets of learners' problem-solving trajectories. The pseudo code of the PPG (Problem-Solving Path Graph) generation algorithm is presented in Figure 2.

Algorithm 1: PPG (*trajectory*, *count*)

input: path data *trajectory*, path frequency data *count*

output: problem solving path graph *G*

```
1 G  // initialize direct graph
2 for i to trajectory
3    for j=0 to length(i)
4       if G.has-edge(j, j+1)
5          G.edge(j, j+1).weight += i.count
6       else
7          G.add-node(j, j+1)
8          G.edge(j, j+1).weight = i.count
9 return G
```

Figure 2. The pseudo code of the PPG algorithm

Table 2 represent the result of PPG algorithm. The vertex in the graph shows different response blocks submitted by learners to solve problems. The number of vertexes in the Hoc 18 mission is larger than that in the Hoc 4 mission since the former includes repetitions - blocks inside blocks - and conditional blocks. The number of edges in Hoc

18 is 4.77 times larger than that of edges in Hoc 4. This indicates that learners use more trajectories in order to solve problems.

Table 2. Generated Problem-Solving Path Graph

Item	Hoc 4	Hoc 18
No. of Vertexes	8,604	68,716
No. of Edges	44,136	210,597

4. The algorithm to generate optimal problem-solving paths

The SPMF (Single Path Maximum Flow) algorithm is designed to find optimal problem-solving paths. Such paths are used by the largest number of learners who solve problems. The Ford-Fulkerson Method is similar to this algorithm in that it calculates the maximum rate at which substances can be transported from the departure point to the destination without exceeding the capacity limit[9]. The SPMF algorithm was developed in this study, and its pseudo code is presented in Figure 5. The algorithm inputs are the problem-solving path graph (G), graph weight (w), and arrival vertex (d). The return value of the algorithm is SPMF tree (T).

Algorithm 2: SPMF (*G*, *w*, *d*)

input: problem-solving path graph *G*, weight *w*, destination vertex *d*

output: maximum flow tree *T*

```
1 T ← G
2 Sort G.E in order that w does not increase
3 for Sorted each edge w(x, y) ∈ E[G]
4   Remove-Edge(T, (x, y))
5   for each v∈ V[T]
6     if not Has-path(T, v, d)
7       Insert-edge(T, (x, y))
8 return T
```

Figure 3. The pseudo code of the SPMF algorithm

Since a trajectory extracted through the SPMF algorithm has the form of a tree, the number of edges is one less than the number of vertexes. Each vertex indicates the level of individual learners. In other words, the number of graded trajectories that can be submitted for the correct answer is 6,547 in Hoc 4 and 37,098 in Hoc 18.

Table 3. Generated Problem-solving Trajectory Tree

Item		Hoc 4	Hoc 18
Trajectory Tree	No. of Vertexes	6,547	37,098
	No. of Edges	6,546	37,097

5. Conclusion

For programming education, individualized education is needed. To do this, it is necessary to develop an algorithm that generates a learning path suitable for the learner level. In this study, the algorithm was developed based on the theory of social constructivism.

The present study proposes a method to provide learners with individualized and customized education in a programming education environment. It seeks to provide learners with proper scaffoldings by referring to the problem-solving trajectories of exemplary companion learners. To this end, graded problem-solving trajectories of learners are extracted from various trajectory sources. The problem-solving trajectories of existing learners are converted into problem-solving graphs. Such problem-solving graphs are applied to the SPMF algorithm. This process was applied in this study and graded problem-solving trajectories were extracted based on the graded problem-solving tree. The SPMF algorithm used in this study extracts trajectories that learners of the same age use to solve problems.

This study proposes a method to extract the problem-solving trajectories of exemplary companions with reference to learners' problem-solving data. In addition, the proposed method provides scaffoldings according to the problem-solving level of each learner based on such trajectories. It is expected that this method can be used as a basis for big data utilization in programming education.

References

[1] C. Piech, M. Sahami, J. Huang, and L. Guibas. Autonomously generating hints by inferring problem solving policies. In Proceedings of the Second (2015) ACM Conference on Learning@ Scale, **(2015)**

[2] Code.org: Anybody can Learn. [online] Available at: https://code.org/ [Accessed 30 Jan. **2018**]

[3] D. Wood, J. S. Bruner, and G. Ross. The role of tutoring in problem solving. J. Child Psych. Psychiatry, 17, 2 **(1976)**

[4] L. E. Berk and A. Winsler. Scaffolding Children's Learning: Vygotsky and Early Childhood Education. NAEYC Research into Practice Series. Volume 7. National Association for the Education of Young Children, Washington, DC **(1995)**

[5] L. S. Vygotsky. Mind in society: The development of higher psychological processes. Harvard University Press **(1980)**

[6] M. Homer and J. Noble. Lessons in combining block-based and textual Programming. J. Vis. Lang. Sent. Sys. 3, 1 **(2017)**

[7] Playentry. [online] Available at: https://playentry.org [Accessed 30 Jan. **2018**]

[8] S. Bocconi, A. Chioccariello, G. Dettori, A. Ferrari, and K. Engelhardt. Developing computational thinking in compulsory education: Implications for policy and practice. EUR 28295 EN;doi:10.2791/792158 **(2016)**

[9] T. Cormen, C. E. Leiserson, R. Rivest, and C. Stein. Instruction to algorithms. MIT Press **(1990)**

