A Tree-Based Chart for Visualizing Programming for Problem Solving

Po-Yao Chao, Yuan Zu University, Taiwan
Yu-Ju Chen, Yuan Zu University, Taiwan

The Asian Conference on Technology in the Classroom 2017
Official Conference Proceedings

Abstract
Programming for solving problems has been an important skill in computer programming education. However, most of the assessment for the skill tends to emphasize on the product of programming rather than the process of programming. Considering students' process of programming may gain insight into the understanding of students' difficulties and their performance, this study incorporates problem solving and visual programming activities to develop a programming learning environment where students interact with the learning environment to solve computational problems. By examining students' behaviors and strategies of problem solving exhibited in the environment, the process and product of students programming activities can be visualized with a tree-based chart. The features and patterns of the tree-based chart may indicate different combination of programming strategies and their effects on performance of problem solving. A case study was conducted to explore the patterns of the tree-based chart. The findings show that the patterns of the tree-based chart were categorized into three different types: accuracy, trial-in-error, and revision. The follow-up interviews were conducted to explore the relationships between the patterns, personal factors, and performance of problem solving.

Keywords: visual programming, tree-based chart, problem solving

iafor
The International Academic Forum
www.iafor.org

Introduction

With the advancement of computer technology, the computational thinking is more and more important (Grover & Pea, 2012). Learning programming is not only a skill but also can help individual improve their reasoning. Due to the importance of computational thinking, many counties advocate for the promotion of programming education in K-12 and childhood education. Moreover, it is believed that computational thinking is related with the development of logical thinking and creativity (Sáez-López, Román-González & Vázquez-Cano, 2016). In Liao and Bright (1991) meta-analysis, they analyzed 65 research about the computational thinking studies. Fifty-eight studies (89%) demonstrated that the computational thinking has a positive impact on the development of children's thinking abilities. Computational thinking is an essential skill for the 21st Century (Einhorn, 2011).

Learning programming is not easy for students because the programming is essentially a problem solving activity (Areias & Mendes, 2007). Programming requires not only basic knowledge of programming concepts and skills but also the ability of problem solving (Linn, 1985). Therefore, assisting students to learn how to formulate problems, analyze problems, and design workable solutions to the problem is important to programming learning. Because the concepts of programming is abstract (Muller & Haberman, 2008) and the syntax of programming language often involves complicated logics, novice learners hardly success in programming (Wilson & Moffat, 2010). Considering learning programming involves understanding abstract concepts, visualization of these abstract concepts may serve as a important tool in learning computational science (Brodlie et al., 1993). Since visualized programming learning environments may offer concrete and intuitive information, programming in these environments may help students comprehend abstract concepts and realize complicated logics when compared with traditional programming languages (Sáez-López et al., 2016).

Assessment is a process that uses information gathered through measurement to analyze or judge a learner's performance on some relevant work task. (Sarkees-Wircenski & Scott, 1995). The traditional programming education tend to more concentrate on the products of programming rather than on the process of programming. Considering students often exhibits strategies of problem solving in the process of solving programming problems, including measurement relevant to the process of programming may improve accuracy of assessment for students' programming abilities. Therefore, in this study, we develop a programming learning environment where students interact with the learning environment to solve computational problems. The environment containing a robot character with which students need to instruct the robot to solve computational problems with pre-defined graphical instructions. Students' behaviors and their use of graphical instructions were visualized and logged for further analysis. The logged data were employed to explore different dimension about the process of solving computational problems. For more detailed address in the state of problem solving, Gagne and Yekovich (1993) defined three different states of problem solving: starting, intermediate, and goal. In this study, the three different states were employed to depict students' process of problem solving in a tree-based chart. The chart aims to reveal different combinations of programming strategies and their corresponding effects on problem solving.

Method

To explore students' behaviors and strategies of problem solving, a case study was conducted to explore the patterns of problem solving in a visualized programming learning environment. The learning environment includes a computational problems and a set of pre-defined graphical blocks. As shown in Figure 1, to the right the graphical representation of computational problems. Students were asked to collect flowers or fruits with limited amount of graphical blocks. Students first create a base to edit instructions to a robot character, and then they can drag and drop graphical blocks to compose programs to instruct the robot to solve computational problem. The programming concepts needed to know for students to solve the problem include sequence, operator, conditional, loop, and variable.

Ten participants were asked to participate in the case study. They were classified in two groups, one is who have programming experience and the other is novice of programming. Participants were asked to solve two parts of computational problems: training and basic. At the beginning, the participants solve training problems that would help them get familiarized with the environment and the function of graphical blocks. After they finished the training problems, the researcher would verified that all the participants did not have problems in solving problems in the visualized programming learning environment. The participants were asked to solve basic problems that were harder and more complex when compared with training problems. Once a participant completed all the basic problems, he/she was given an interview. The interviews were recorded to explore different combination of programming strategies and the relationship between tree-based chart and user's performance.

We use the tree-based chart to visualize the process of the problem solving. The features of the tree-based chart reveal the pattern of problem solving. The features include depth, node number, density, instruction accuracy. The depth is the number of segmentation of the problem. The node number is the number of execution. The density is the accuracy of the problem solving. The instruction accuracy is the ratio of insert instruction and update instruction. The higher the value, the more accurate.

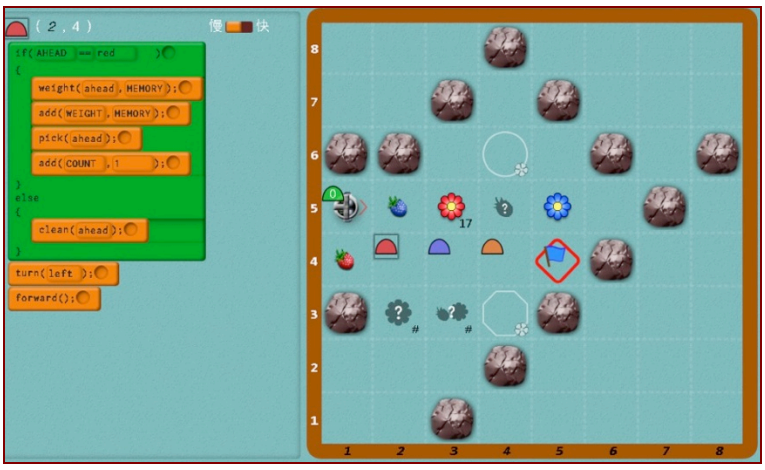

Figure 1: Example of Computational Problems.

Result

10 participants (4 competent users and 6 novices) in the case study, age about 20 to 30 years old. According to the features and pattern of the tree-based chart, the patters were categorized into three types: accuracy, revision, and trial-and-error (Figure 2). Different color of tree node in the tree-based charts represents different states of problem solving. The green, red, and blue node represent starting, intermediate, and goal state. When users execute the instruction of a base, the execution creates a new node in next layer representing the new state of the execution. Participants could back trace to the previous base on the map, which changes the state of problem solving.

Tree type	Depth	Node Number	Density	Instruction Accuracy
Accuracy	4	4	1	100%
Revision	5	7.5	1.5	91%
Trial and error	7.5	14.1	1.89	68%

Table 1: Participants' averages of problem segmentation, execution, and instruction accuracy in solving a computational problem.

Table 1 shows participants' averages of problem segmentation, execution, and instruction accuracy in solving a computational problem (Figure 1). Participants were divided into three groups: accuracy, revision, and trial-and-error. The accuracy group shows low density and high instruction accuracy. However, the trial-and-error group reveals high density and low instruction accuracy. The revision group demonstrates medium density and high instruction accuracy. To visualized participants' patterns of solving problems, the abovementioned indicators were transformed into tree-based charts (Figure 2). The accuracy tree-based chart features almost no branch, all degree is 1. The revision tree-based chart has small amount of branches around 2 to 3. Finally, the trial-and-error tree-based chart has large amount of branches.

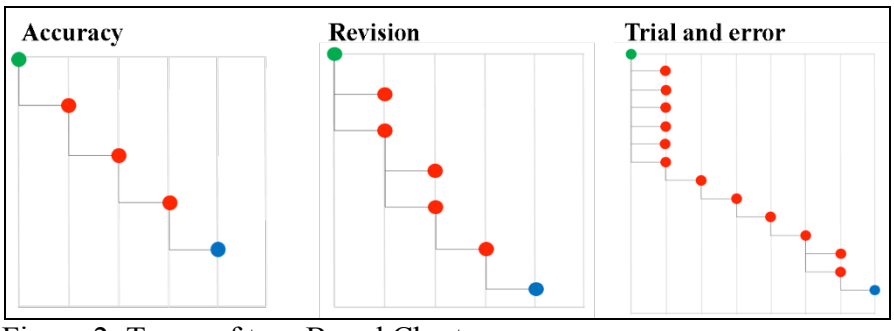

Figure 2: Types of tree-Based Charts

In the interview, we compared participants' problem solving behaviors between competent and novice students. In terms of problem segmentation, competent students tend to complete a series of steps, and then create a new base to solve the rest parts of the problem. For example, they pick up all of the flowers, then create a base to clean the grass. However, the novices tend to make many bases, because they want to minimize the number of instructions for each base. In terms of loop use, if competent users find something needs to repeat many times, they will use the loop to complete the problem. On the other hand, the novices seldom use the loop at the beginning. They will not use loop blocks until the amount of graphical blocks exceed the limit.

For debugging, competent users generally applied breakpoints to observe robot's behavior and perceived the breakpoints very useful for debugging. The novices rarely used breakpoint functions but always try many time to find the errors.

Though the interview and data, we can explore the relationships between the patterns, personal factors, and performance of problem solving. Some of the competent users tend to demonstrate accuracy features. They always had good strategies in the problem, so they seldom encounter mistakes. Their tree will like the step that was no branch. Their value of instruction accuracy was very high. Some of the competent users tend to exhibits revision features. When their instruction occurred an error, they would use breakpoints to debug. They can find error accurately, so they did not execute many times to find an error. Their tree would have some branch but not too more. Novices tend to trial and error. They executed many times in same instruction or made a lot of bases. So, the tree of novices will produce many nodes in the same layer or more depth than competent users.

Conclusion

The visualize programming learning environment seems effective to help learner learning computational thinking and promote their motivation to solve problems. Many users, especially novices, who think the environment allows him/her to learn the basic concepts of programming.

In this study, by examining students' behaviors and strategies of problem solving exhibited in the environment, the process and product of students programming activities can be visualized with a tree-based chart. The chart is easier to observe students' process of problem solving. For example, a tree has many nodes in the same layer may reveal that a student may have obstacle in understanding programming concepts when solving computational problems. The large amount of program execution may demonstrate that the student try and error many times. The features and the patterns of the tree-based charts may help teachers or students assess the performance of solving computational problems, which may gain insight into students' process of programming for solving problems.

References

Areias, C., & Mendes, A. (2007). A tool to help students to develop programming skills. *Proceedings of International Conference on Computer Systems and Technologies*, 20:1-20:7.

Brodlie, Ie, Poon, A., Wright, H., Brankin, L., Banecki, G., & Gay, A. (1993). GRASPARC - A problem solving environment integrating computation and visualization., *In Proceedings of the IEEE Conference on Visualization*, 102-109.

Einhorn, S. (2011). Microworlds, computational thinking, and 21st century learning: Logo Computer System Inc, White Paper.

Gagne, E. D., Yekovich, C. W., & Yekovich, F. R. (1993). *The cognitive psychology of school learning.* New York, NY: HarperCollins College Publishers.

Grover, S., & Pea, R. (2012). Computational thinking in K-12: a review of the state of the field. *Educational Researcher*, 42(1), 38-43.

Muller, O., & Haberman, B. (2008). Supporting abstraction processes in problem solving through pattern-oriented instruction. *Computer Science Education*, 18(3), 187–212.

Liao, Y. C., & Bright, G. W. (1991). Effects of computer programming on cognitive outcomes: a meta-analysis. *Journal of Educational Computing Research*, 7(3), 251–266.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher*, 14(5), 14-29.

Robins, A., Rountree, J., & Rountree, N. (2003). Learing and teaching programming: A review and discussion. *Computer Science Education*, 13, 137-172.

Sáez-López, J., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using 'Scratch' in five schools. *Computers & Education*, 97, 129-141.

Sarkees-Wircenski, M., & Scott, J. L. (1995). *Vocational Special Needs*. Homewood, IL: American Technical Publishers.

Wilson, A., & Moffat, D. C. (2010). Evaluating Scratch to introduce younger school children to programming. In *Proceedings of the 22nd Annual Psychology of Programming Interest Group (Universidad Carlos III de Madrid, Leganes, Spain).*