ISBN: 978-989-8533-43-2 © 2015

A PROGRAM COMPLEXITY METRIC BASED ON VARIABLE USAGE FOR ALGORITHMIC THINKING EDUCATION OF NOVICE LEARNERS

Minori Fuwa[1], Mizue Kayama[2], Hisayoshi Kunimune[2], Masami Hashimoto[2] and David K. Asano[2]
[1]*Graduate School of Science and Technology, Shinshu University, Japan*
[2]*Faculty of Engineeri , Shinshu University, Japan*

ABSTRACT

We have explored educational methods for algorithmic thinking for novices and implemented a block programming editor and a simple learning management system. In this paper, we propose a program/algorithm complexity metric specified for novice learners. This metric is based on the variable usage in arithmetic and relational formulas in learner's algorithms. To evaluate the applicability of this metric for novice education, we discuss the differences between three previous program complexity metrics and our proposed metric.

KEYWORDS

Algorithmic thinking, program complexity metric, novice education, variable usage, educational aspect

1. INTRODUCTION

Not only in undergraduate schools but also in pre-university education, 21st century skills are some of the most important learning topics [1,2]. In 2009, an OECD report pointed out three ICT related competences for new millennium learners [3]. 21st century skills were one of them. 21st century skills are a framework containing some skills, abilities and knowledge related to ICT. Recently, Informatics/Computing is one of the main subjects in junior or senior high school [4]. In these subjects, algorithms and algorithmic thinking are key topics [5]. However, many tools do not have ways to control the learning process. Also, many tools do not have ways to evaluate and assess student's work.

We proposed an educational method for algorithmic thinking in fundamental education for computer science course students in 2008 [6] and examined the effectiveness of our algorithmic thinking learning support system. This system uses only three flow structure elements: calculation, selection and repetition. Students describe the algorithm using these three elements. To assess learner's understanding, we evaluated each learner's algorithm in detail. As a result, we found eight types of errors in their algorithms. Based on this fact, we think that if an instructor can give an appropriate sub-task to a learner who has misunderstanding or confusion when creating an algorithm, they can revise their mental model or knowledge of algorithm creation. To fulfill this goal, a program/algorithm complexity metric is needed in this study.

This paper describes a program/algorithm complexity metric for our algorithmic thinking course and also discusses some possibilities of our proposed metric as an assessment function for novice learners.

2. OUR APPROACH

We developed a tool called AT to describe algorithms for novice learners. AT can describe the algorithm with block programming. Learners can make ten types of blocks in AT. Three types of blocks (plan, conditional branching and loop) can contain other blocks.

We use three learning items in this class: calculation, conditional branching and loop. In this study, four types of problems are created using a combination of these learning items: calculation problem, conditional branching problem, loop problem and combination problem. The calculation problem uses only calculations. The conditional branching problem uses calculations and conditional branching. The loop problem uses calculations and loops. The combination problem uses all the learning items. In this study, the maximum number of lines in the algorithm is thirty.

The purpose of this study is to explore educational methods for algorithmic thinking conceptual modeling for novices, so we think that it is important to make support problems. Support problems are different from problems for all students who attend our algorithmic thinking class. Support problems are problems that overcome students' weak points. We want all students to understand the algorithms. Therefore, we identify errors in students' answers and teach these points to the students. Then we need to judge their weak points and make support problems to overcome these weak points. To make support problems we need a metric.

There have been proposed some metrics used to evaluate programs. For example, McCabe's cyclomatic complexity metric [7], Halstead's complexity metric [8] and Hakuta's complexity metric [9]. We discuss the problems with these three metrics related to our learning tool. First, the three metrics do not take into account the way the variables are used in calculations and control statements. Second, they can only evaluate algorithms that are complete. Third, they are not suitable when the level of difficulty of the learning item changes. Therefore, we need a new metric to resolve these problems.

In this paper, we write about two points. The first point is to introduce our proposed new metric. The second point is to describe the characteristics and differences between previous metrics and our metric.

3. EDUCATIONAL PROGRAM COMPEXITY METRIC FOR NOVICE LEARNERS: FL

We propose a new metric to solve the problems with the three previous metrics. In this section we describe the characteristics and evaluation method of our metric.

3.1 Evaluation Method

The metric proposed in this study is called "Educational program complexity metric for novice learners" and its abbreviated designation is FL. In this study, calculations are limited to dyadic operations. Therefore, the number of operands and arithmetic operators and the kinds of variables are decided in calculations. Also, conditional expressions in control statements are limited to two operands and one comparison operator. We take into account the way operands and operators are used in calculations and conditional expressions. We also consider nested control statements.

Table.1-3 shows the way we evaluate each learning item. Table.1 shows the levels for calculations. There are six calculation levels defined according to the number of operands and the kind of variables. Table.2 shows the levels for conditional branching, while Table.3 shows the levels for loops. There are two conditional branching levels and two loop levels defined according to the variable or number used in the comparison in the conditional expression.

Table 1. Calculation levels

CL	Summary	Examples
1	Only numbers	a = 0 a = 1 + 2
2	Different variable and a number	a = b a = b + 1
3	Same variable and a number	a = a + 1
4	Two variables	a = b + c
5	Same variable and another variable	a = a + b
6	Same variables	a = a + a

Table 2. Conditional Branching levels

CBL	Summary	Example
1	Compare with a number	if (a < 1)
2	Compare with a variable	if (a < b)

Table 3. Loop levels

LL	Summary	Example
1	Compare with a number	while (a < 1)
2	Compare with a variable	while (a < b)

The features of our method to evaluate an algorithm are shown here. First, a level is assigned to each line in the algorithm using Table1-3. The levels are denoted by CL, CBL and LL. Second, if each learning item is not a line, the levels of each learning item are added. We call these sums S_{CL}, S_{CBL} and S_{LL}. Third, the levels of the three learning items are added. Finally, if there is a nested control statement, their number is added to the three learning items' levels. We name this value R_{EA}. The upper parts of Figure 1 and Figure 2 show two sample algorithms and the values that result from FL. The left algorithm is a calculation problem and the right algorithm is a combination problem. The calculation problem is to calculate the BMI with two variables (hei and wei) as input numbers. The combination problem is to calculate the product of two variables (x and y) for values of x and y from 1 to 9. If the product is divisible by two, the product is output. Sample calculations of FL are shown for the two algorithms in the upper parts of Figure 1 and Figure 2 in the lower parts of both Figure 1 and Figure 2. From this result, the R_{EA} of the combination problem is higher than the calculation problem, so the combination problem is more complex than the calculation problem.

<Calculation problem>

```
1 hei = 0;                    CL 1
2 wei = 0;                    CL 1
3 bmi = 0;                    CL 1
#
4 input (hei);
5 input (wei);
#
6 hei = hei * hei;            CL 6
7 bmi = wei / hei;            CL 4
#
8   output (bmi);
```

$S_{CL} = 1 + 1 + 1 + 6 + 4 = 13$
$S_{CBL} = 0$
$S_{LL} = 0$
$N_{ncs} = 0$

$R_{EA} = 13 + 0 + 0 + 0 = 13$

Figure 1. Result of calculation

<Combination problem>

```
1 x = 1;                      CL 1
2 z = 0;                      CL 1
3 z2 = 0;                     CL 1
#
4 while (x < 10) {            LL 1
5  y = 1;                     CL 1
6  while (y < 10) {           LL 1
7    z = x * y;               CL 4
8    z2 = z mod 2;            CL 2
9    if (z2 == 0) {           CBL 1
10     output (z);
     }
11   y = y + 1;               CL 3
   }
12  x = x + 1;                CL 3
  }
```

$S_{CL} = 1 + 1 + 1 + 1 + 4 + 2 + 3 + 3 = 16$
$S_{CBL} = 1$
$S_{LL} = 1 + 1 = 2$
$N_{ncs} = 2$

$R_{EA} = 16 + 1 + 2 + 2 = 21$

Figure 2. Result of combination

We take into account the way operands are used in formulas and conditional expressions in control statements. Because FL sets levels for each learning item, we can evaluate each line in an algorithm. Therefore, we can evaluate incomplete algorithms. Also, because FL can transcribe error algorithms, it can make support problems in this study.

3.2 Comparison of Result between Previous Metrics and FL

Table.4 shows the characteristics of the three previous metrics and FL. There are two points of view and seven items. The two points of view are structural point of view and educational point of view. The structural point of view looks at the structure of algorithms. The educational point of view is concerned with support problems.

Metrics that have the characteristics (a)-(g) have a ✔ in the corresponding column in Table.4. First, McCabe's cyclomatic complexity metric cannot evaluate algorithms from a structural point of view and educational point of view in this study. Second, from a structural point of view, Halstead's complexity metric can evaluate operands and operators in an algorithm. However, it cannot evaluate nested control statements. From an educational point of view, it is impossible to change algorithms, so (e)-(g) do not apply to this metric. Third, from a structural point of view, Hakuta's complexity metric can evaluate algorithms because there are measures that evaluate operands, operators and algorithm length. From an educational point of view, the metric also does not apply to (e)-(g). Finally, FL sets levels in detail according to operands and operators. Therefore FL can evaluate algorithms from a structural point of view. Also, by adding the number of nested control statements, FL can evaluate them. Therefore, FL can evaluate algorithms like previous

metrics. From an educational point of view, FL sets levels to evaluate each line and each learning item, so all items apply. Therefore, FL is the only metric that can be used to make support problems.

Table 4. Characteristic of previous metrics and FL

Evaluation points		Previous metrics			FL
		McCabe	Halstead	Hakuta	
Structural points	(a) Uses the number of variable types		✔	✔	✔
	(b) Uses the number of variables and numbers		✔	✔	✔
	(c) Metric changes with program length		✔	✔	✔
	(d) Considers nested control statements: there are two types of algorithms			✔	✔
Educational points	(e)Designed for use on unfinished programs				✔
	(f) Can evaluate program subsets: to make support problems				✔
	(g) Can evaluate different learning item levels				✔

The purpose of this study is to explore educational methods for algorithmic thinking conceptual modeling for novices. Therefore, it is necessary to make support problems. Support problems are made based on errors in students' solutions, so it is necessary to change the levels of problems. From Table.4, all items apply to FL. Also, four items apply to Halstead's complexity metric from a structural point of view. Therefore, we use FL to make support problems and use Halstead's complexity metric to choose support problems suitable for each student.

4. CONCLUSION

In this paper, we proposed a program complexity metric based on variable usage for algorithmic thinking education. The features and problems of three previous metrics were discussed. The main problems with previous metrics were the three metrics do not take into account the way the variables are used in calculations and control statements. To avoid these problems, we developed a new metric for novice learners' algorithmic thinking education. The characteristic of our proposal is that each learning item can be evaluated. Moreover, unlike previous metrics, our metric can evaluate each line in an algorithm. Therefore, we decided to use FL to make support problems for novice learners who submit algorithms with errors.

We think this metric is suitable for evaluating novices' algorithms. However, we did not evaluate the effectiveness of our metric in real educational situations. In the future, we will make rules to make support problems. Using these rules, we will make support problems according to the error levels in students' solutions. After giving the support problems to students, we will check whether students' understanding improves.

ACKNOWLEDGEMENT

This study was supported by JSPS Grant_in_Aid for Scientific Research (B) Grant Number 22300286.

REFERENCES

Partnership for 21st century skills, http://www.p21.org/ (accessed 2015/7/27).

K. Kärkkäinen, 2012. Bringing About Curriculum Innovations Implicit Approaches in the OECD Area. *OECD Education Working Papers*, No.82.

K. Ananiadou. et al, 2009. 21st century skills and competencies for new millennium learners in OECD countries. *OECD Education Working Paper*, No.41.

S.P. Jones. et al, 2011. "Computing at School International comparisons," http://www.computingatschool.org.uk/data/uploads/internationalcomparisons-v5.pdf (accessed 2015/7/27).

The CSTA Curriculum Improvement Task Force, 2008. "The New Educational Imperative: Improving High School Computer Science Education *International Version*," http://www.csta.acm.org/Communications/sub/DocsPresentationFiles/NewImperativeIntl.pdf (accessed 2015/7/27).

M. Kayama. et al, 2014. Algorithmic Thinking Learning Support System Based on Student-Problem Score Table Analysis. *Int. J. of Computer and Communication Engineering*, Vol.3, No.2, pp.134-140.

T.J. McCabe, 1976. A Complexity Measure. *IEEE Transactions on Software Engineering*, Vol.2, No.4, pp.308-320.

M.H. Hasltead, 1977. *Elements of Software Science*. Elsevier North-Holland, NY.

M. Hakuta, 1995. A Study of Software Metrics and Productivity Evaluation. *IEICE Trans. of Information Systems*, Vol.J78-D-I, No.12, pp.936-944.