Assessing Your Algorithm: A Program Complexity Metrics for Basic Algorithmic Thinking Education

Mizue Kayam*
Shinshu University
Nagano, Japan
kayama@cs.shinshu-u.ac.jp

Minori Fuwa
Graduate School of Shinshu University
Nagano, Japan

Hisayoshi Kunimune
Shinshu University
Nagano, Japan

Masami Hashimoto
Shinshu University
Nagano, Japan

David. K. Asano
Shinshu University
Nagano, Japan

***Abstract*— We have explored educational methods for algorithmic thinking conceptual modeling for novices and implemented a block programming editor and a simple learning management system. This system has been used in our algorithmic thinking course since 2008. Based on this experience, 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. Moreover, 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 assessment

I. 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. This framework and the learning objectives of Informatics [4] (in EU countries), Computing [5] (in the UK) or Computer Science (in the US) [6] are strongly connected [7]. Recently, Informatics/Computing is one of the main subjects in junior or senior high school [8]. In these subjects, algorithms and algorithmic thinking are key topics [9-14].

There have been many papers on this kind of algorithm and/or algorithmic thinking education, e.g., [15-17]. In some research, general programming tools like Java, g++, Scratch [18], BYOB [19] and Squeak & Alice [20] are used as the learning environment. In others, some original learning tools are used to help students express their algorithms. Some tools visualize the student's algorithm, and others animate the student's work.

However, many tools do not have ways to control the learning process, i.e., the teacher cannot customize the tool to suit his/her lesson objectives. Also, many tools do not have ways to evaluate and assess student's work.

In our research, algorithmic thinking is defined as a thinking method to exchange personal ideas about problem solving with others. Our subjects are mainly university freshmen and high school students. To foster the formation of mental images of algorithms and data processing, our students should take the algorithmic thinking course before they start to learn programming. We focus on algorithmic thinking as the first step for students in our department. After this experience, they will start to learn about programming and data structures using a programming language like C, C++ or Java.

We proposed an educational method for algorithmic thinking in fundamental education for computer science course students in 2008 [21,22] and examined the effectiveness of our algorithmic thinking learning support system. We designed our system based on discovery learning [23] and guided discovery [24]. 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.

II. Our Approach

A. Previous Progress

We developed a tool called AT to describe algorithms for novice learners [25]. Figure 1 shows an algorithm using AT. 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. Learners can save unfinished algorithms in AT. When they finish writing the algorithm, they submit it with AT. A teacher evaluates the learner's algorithm. This evaluation can be seen by the learner.

Because our subjects are novice learners, only integer variables are used and only one or two-dimensional arrays can be processed. Therefore, the input data type is "variable". However, the output data types are "variable" and "string". Some kinds of messages can be used in our learner's algorithm. Calculations are limited to binary operations. The operators that can be used are the five arithmetic operations (+, -, *, / and % [remainder]) and the relational operations (<, <=, >, >=, ==, !=). The flow structures are "selection" and "repetition". As for repetition (iteration) operations, two types of elements: iteration with loop time (for type) and iteration with exit/continue condition (while type) are provided.

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.

B. This Report

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.

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

III. Program Complexity Metrics

In this section, we introduce three previous metrics used to evaluate programs and show evaluation results when these metrics are applied to the sample problems.

A. McCabe's Cyclomatic Complexity Metric

McCabe's cyclomatic complexity metric was proposed by T.J.McCabe in 1976 [26]. The complexity of a program is measured by the number of control statements used. In this study, the metric is calculated by adding 1 to the number of control statements. This metric has a positive correlation with the number of bugs in a program. A variable and formula for this metric are shown below.

$$M = N_{sc} + 1 \qquad (1)$$

N_{sc} : Number of control statements
M : McCabe's cyclomatic complexity metric

From these results, we see that the complexity increases as the number of control statements increases.

B. Halstead's Complexity Metric

Halstead's complexity metric was proposed by M.H.Halstead in 1977 [27]. When a program gets longer, the numbers of operators and operands increases. This indicates that a program is more complex.

This metric has some parameters. In this paper, we use the parameters: program vocabulary, program length, volume, difficulty and effort. Variables and formulas used to calculate this metric are shown below (rectangle part). The items we use are η, N, V, D and E.

η : program vocabulary

$$\eta = \eta_1 + \eta_2 \qquad (2)$$

N : program length

$$N = N_1 + N_2 \qquad (3)$$

V : volume

$$V = N \times \log_2\eta \qquad (4)$$

D : difficulty

$$D = (\eta_1 / 2) \times (N_2 / \eta_2) \qquad (5)$$

E : effort

$$E = D \times V \qquad (6)$$

η_1 : number of unique operators
η_2 : number of unique operands
N_1 : total number of operators
N_2 : total number of operands

C. Hakuta's Complexity Metric

Hakuta's complexity metric was proposed by M.Hakuta in 1995[28]. It measures the complexity of a program written in C. It has three points: lexical complexity, structural complexity and logical complexity.

First, lexical complexity is calculated from the size of the program and the vocabulary. Second, structural complexity is calculated from the depth of the program in terms of the number of nested control statements. Finally, logical complexity is calculated from the logical difficulty of the processing. Because we do not use functions in this study, we only use lexical complexity and structural complexity. Lexical complexity has four parameters: program scale, program vocabulary, operation degree and data handling degree. Because we are interested in the number of variables and numbers in this study, we use program scale and program vocabulary to measure lexical complexity.

Variables and formulas used to calculate this metric are shown below (rectangle part). The items we use are P_s, P_v and μ_s.

P_s : Program scale

$$P_s = N_s + S_v \tag{7}$$

P_v : Program vocabulary

$$P_v = N_v + N_p \tag{8}$$

μ_s : Structural complexity

$$\mu_s = \frac{1}{N_s} \left\{ 1 + \left(\frac{N_{sn}}{N_{sc}} \right)^2 \right\} N_{sc} \tag{9}$$

N_s : Number of statements
S_v : Kind of variable
N_v : Number of variables and constants
N_p : Number of operators
N_{sc} : Number of control statements
N_{sn} : Number of nested control statements

When an algorithm gets longer, the algorithm length, the number of operators and the number of operands increase. This results in a larger program scale and program vocabulary.

Structural complexity uses the number of control statements and nested control statements. When the number of control statements and the number of nested control statements are almost the same, the structure is deeper, so the structural complexity is higher.

D. *Problems with the Previous Metrics*

Here 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.

IV. 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.

A. *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 [29].

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 IV. 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 V. CONDITIONAL BRANCHING LEVELS

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

TABLE VI. LOOP LEVELS

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

The method to evaluate an algorithm is shown below.

R_{EA}: Result of evaluated algorithm
$R_{EA} = S_{CL} + S_{IL} + S_{WL} + N_{sn}$
S_{CL}: Sum of calculation levels (CL)
S_{CBL}: Sum of conditional branching levels (CBL)
S_{LL}: Sum of loop levels (LL)
N_{sn}: Number of nested control statements

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, add the levels of each learning item. We call these sums SCL, SCBL and SLL. Third, add the levels of the three learning items. Finally, if there is a nested control statement, add their number to the three learning items' levels. We name this value REA. Figure 1 and Figure 2 show the level assignments for the two sample problems.

```
<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);
```

Fig. 1. Result of calculation problem.

```
           <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
  }
```

Fig. 2. Result of combination problem.

B. Features

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.

C. 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. It has items (a)-(d). (a) Uses the number of variable types: when the number of variable types increases in an algorithm, an algorithm is more complex. (b) Uses the number of variables and numbers: calculations are limited to dyadic operations. Therefore, when the number of operands increases in an algorithm, an algorithm is more complex. (c) Metric changes with program length: when the number of calculations increases in an algorithm, the algorithm length increases, resulting in a change in the algorithm difficulty. (d) Considers nested control statements: there are two types of algorithms: ones that use nested control statements and ones that do not. We assume that an algorithm with nested control statements is more complex.

The educational point of view is concerned with support problems. (e) Designed for use on unfinished programs: we assume that there are students who submit unfinished algorithms. We think that it is necessary to evaluate them. (f) Can evaluate program subsets: to make support problems, it is necessary to discover errors and evaluate their levels. (g) Can evaluate different learning item levels: to make support problems, it is necessary to change the levels of the three learning items.

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.

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,

TABLE IV. 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				?

we use FL to make support problems and use Halstead's complexity metric to choose support problems suitable for each student.

V. Conclusion

In this paper, we proposed a program complexity metric based on variable usage for algorithmic thinking education. At first, three previous metrics to measure program complexity were introduced. The features and problems of these 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.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 22300286 & 16H03074.

References

[1] Partnership for 21st century skills, http://www.p21.org/ (accessed 2016/6/15).

[2] K.Kärkkäinen, 2012. Bringing About Curriculum Innovations Implicit Approaches in the OECD Area. OECD Education Working Papers, no.82.

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

[4] R.T.Mittermeir, 2005. From Computer Literacy to Informatics Fundamentals, Proc. of International Conference on Informatics in Secondary Schools -Evolution and Perspectives.

[5] UK Department of Education, 2013. "National curriculum in England: computing programmes of study (in HTML),".

[6] ACM, 2003. "A Model Curriculum for K–12 Computer Science: Final Report,".

[7] V. Dagienė, 2011. Informatics Education for New Millennium Learners. Proc. of 5th International Conference on Informatics in Secondary Schools -Evolution and Perspectives, pp.9-20.

[8] S.P. Jones. et al, 2011. Computing at School International comparisons, http://www.computingatschool.org.uk/data/uploads/internationalcomparisons-v5.pdf (accessed 2016/6/15).

[9] The College Board, AP Computer Science Principles. https://advancesinap.collegeboard.org/stem/computer-science-principles/ (accessed 2016/6/15).

[10] Computing at School, http://www.computingatschool.org.uk/ (accessed 2016/6/15).

[11] Ontario Ministry of Education, The Ontario Curriculum, http://www.edu.gov.on.ca/eng/curriculum/secondary/subjects.html/ (accessed 2016/6/15).

[12] Republic of South Africa Department of Basic Education, FET Curriculum Assessment Policy Document, http://www.education.gov.za/Curriculum/CurriculumAssessmentPolicyStatements/CAPSFETPhase/tabid/420/Default.aspx/ (accessed 2016/6/15).

[13] O.Hazzan. et al, 2008. A model for high school computer science education: The four key elements that make it! Proc. of 39th SIGCSE technical symposium, pp.281-285.

[14] The CSTA Curriculum Improvement Task Force, 2008. The New Educational Imperative: Improving High School Computer Science Education *International Version*.

[15] N.Sarawagi, 2010. A general education course - Introduction to Algorithmic Thinking - using Visual Logic. Journal of Computing in Sciences in Colleges, Vol.25, No.6, pp.250-252.

[16] S.Hubalovsky. et al, 2010. Modeling of a real situation as a method of the algorithmic thinking development and recursively given sequences. WSEAS Transactions on Information Science and Applications, Vol.7, No.8, pp.1090-1100.

[17] G.Futschek. et al, 2011. Learning Algorithmic Thinking with Tangible Objects Eases Transition to Computer Programming. Proc. of 5th International Conference on Informatics in Secondary Schools: Evolution and Perspectives, pp.155-163.

[18] J.Maloney. et al, 2004. Scratch: A Sneak Preview. Proc. of Second International Conference on Creating, Connecting and Collaborating through Computing, pp.104-109.

[19] B.Harvey. et al, Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?, Proc. of the Constructionism 2012, http://www.eecs.berkeley.edu/bh/BYOB.pdf (accessed 2016/6/15).

[20] S.Cooper. et al, Developing Algorithmic Thinking With Alice, Proc. of the Information Systems Education Conference 2000, http://web.stanford.edu/~coopers/alice/isecon00.PDF (accessed 2016/6/15).

[21] Y.Fuwa. et al, 2009. Implementation and Evaluation of Education for Developing Algorithmic Thinking for Students in Computer Science. IEICE Technical Report of Education Technology, Vol.109, No.268, pp.51-56.

[22] 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.

[23] J.Bruner, 1960. The Process of Education, Harvard University Press. MA.

[24] M.E.Cook, 1991. Guided Discovery Tutoring: A Framework for Icai Research. Van Nostrand Reinhold, NY.

[25] K.Kobayashi. et al, 2012. Development of a Visual Programming Environment to Support Algorithmic Thinking Education. IEICE Technical Report, Vol.27, No.4, pp.3-8.

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

[27] M.H.Hasltead, 1977. Elements of Software Science. Elsevier North-Holland, NY.

[28] 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.

[29] M.Fuwa. et al, 2014. Basic Consideration for Generating Auxiliary Problems in Algorithm Learning. IEICE Technical Report, Vol.114, No.260, pp.61-66.