RECENT APPROACHES FOR COVERING SOLUTION SPACE OF INTELLIGENT TUTORING SYSTEMS FOR PROGRAMMING

CÁC TIẾP CẬN TRONG QUẢN LÝ KHÔNG GIAN LỜI GIẢI CỦA CÁC HỆ THỐNG DẠY KÈM THÔNG MINH TRONG LĨNH VỰC LẬP TRÌNH

Nguyen Van Y
Ho Chi Minh City Cadre Academy, Vietnam

Received 18/12/2016, Peer reviewed 12/01/2017, Accepted for publication 24/01/2017

ABSTRACT

Introductory programming is an essential part of the curriculum in any engineering disciplines in universities. However, for many beginning students, it is very difficult to learn. In particular, these students often get stuck and frustrated when attempting to solve programming exercises. One way to assist beginning programmers to overcome difficulties in learning to program is to use intelligent tutoring systems (ITSs) for programming, which can provide students with personalized hints of students' solving process in programming exercises. In ITSs for programming, a single programming exercise may produce many alternative solutions from students. It is difficult to build ITSs for programming due to the complexity and variety of possible solutions. In order for the students to learn from the system, it is necessary for them to receive feedback on their solutions to the programming exercises. To provide personalized feedback to students who are solving programming exercises effectively, the ITSs for programming must be able to cover a large space of possible solutions. The goal of this paper is to provide a brief review of the works in the literature related to this problem.

Keywords*: Intelligent Tutoring System; Programming Tutor; Programming Exercises; Solution Space; Feedback.*

TÓM TẮT

Nhập môn kỹ thuật lập trình là một môn học thiết yếu trong chương trình đào tạo của tất cả các ngành kỹ thuật ở bậc đại học. Tuy nhiên, đối với nhiều sinh viên mới vào học, đây lại là môn học khó. Họ thường gặp bế tắc khi giải quyết các bài tập lập trình. Một giải pháp để hỗ trợ sinh viên khắc phục khó khăn này là hệ thống dạy kèm thông minh (ITSs) nhằm cung cấp cho sinh viên các gợi ý trong quá trình làm bài tập. Trong các hệ thống dạy kèm thông minh của lĩnh vực lập trình, một bài toán lập trình đơn giản có thể có nhiều lời giải khác nhau từ các sinh viên. Do vậy, sẽ khó khăn để xây dựng các hệ thống này vì phải làm việc trên một không gian đa dạng các lời giải. Sinh viên khi học từ các hệ thống dạy kèm thông minh của lĩnh vực lập trình, họ cần nhận các phản hồi từ hệ thống về các lời giải của họ. Để cung cấp các phản hồi theo từng trường hợp của sinh viên khi đang giải các bài toán lập trình, các hệ thống dạy kèm thông minh phải có khả năng quản lý một không gian lớn các lời giải có thể có từ các sinh viên. Mục tiêu của bài báo này là giới thiệu một số các giải pháp hiện nay đối với hệ thống dạy kèm thông minh của lĩnh vực lập trình.

Từ khóa: *Hệ thống dạy kèm thông minh; Hệ thống dạy kèm lập trình; Bài tập thực hành lập trình; Không gian lời giải; Phản hồi.*

I. INTRODUCTION

Introductory programming is an essential part of the curriculum in any engineering discipline in universities. However, for many beginning students, it is very difficult to learn. In particular, these students often get stuck and frustrated when attempting to solve programming exercises. One way to assist beginning programmers to overcome difficulties in learning to program is to use ITSs for programming, which can provide students with personalized hints of students' solving process in programming exercises. The major challenge here is that even simple programming exercises can have multiple separate correct solutions and hundreds of intermediate states, and each of these states can be rewritten in hundreds of ways by varying the code's ordering or adding extra code. As an example, a simple programming exercise is given as: "Calculate and print the sum of all odd positive numbers smaller than 100". This programming exercise can be solved in multiple ways using construction of an imperative programming language, for example:

```
// option1
int sum1 = 0;
for (int i = 1;
i < 100; i = i
+ 2)
{
sum1 =
sum1 + i;
}
print(sum1);
```

```
// option2
int sum2 = 0;
for (int i = 1;  i
< 100; i++)
{
if (i % 2 == 1)
{
sum2 = sum2
+ i;
}
}
print(sum2);
```

```
// option3
print(pow(
100/2, 2));
```

We can also think of many variants for any of these solutions, for example:

```
// variant 1
int counter =
1;
int sum1 = 0;
while (counter
<= 100)
{
sum1 +=
counter;
counter +=
2;}print(sum1)
;
```

```
// variant 2
int x = 100
/ 2;
int sum2 =
x * x;
print(sum2)
;
```

```
// variant 3
print(2500)
;
```

In this simple example we can truly see many syntactic differences, such as:

- Using a while loop instead of a FOR loop.
- Using a compound assignment operator (counter += 2) instead writing out the full assignment (counter = counter + 2).
- Using a different name for a variable.

We can also identify a minor semantic difference in variant 1: looping until the counter is at least 101 instead of 100. The result is still a correct program. Also, if we swap two independent statements, do we get a different solution? Another issue is performing a calculation in steps instead of in a single assignment and even only printing the expected end result. Are these different solutions or simply variants of the same solution? [1]

One of the main functions of ITSs for programming is providing feedback to instruct students how to solve programming exercises. Understanding solution variation is important for providing appropriate feedback to students [2]. It is necessary that the system be capable of identifying all such variations [3].

II. BACKGROUND

A variety of ITSs for programming have been built to provide tutoring services for programming problems. In this paper, 51 ITSs for programming (Table 1, Table 2 and Table 3) may be divided into three classes: 1) ITSs for curriculum sequencing (class 1), 2) ITSs for analyzing solution (class 2) and 3) ITSs for programming problem solving support (class 3).

The goal of ITSs for curriculum sequencing is to provide students with the most suitable individually planned sequence of programming concepts/topics to learn and learning tasks (examples, questions, etc) to work with. It helps the students find an “optimal learning path” through the learning material (learning content). In the context of Web-based education, curriculum sequencing technology becomes very important due to its ability to guide the student through the hyperspace of available information. The typical ITSs for programming of this class are No. 12, No. 13 (Table 1), No. 21, No. 22 (Table 2), No. 44 (Table 3). ITSs for analyzing solution deal with students' solutions of programming exercises. Unlike automated grade system which can only tell whether the solution is correct or not, theses ITSs can tell what is wrong or incomplete and which missing or incorrect pieces of knowledge may be responsible for the error(s). These ITSs can provide the student with extensive error feedback. The typical ITSs for programming of the class 2 are No. 4, No. 5, No. 6, No. 7 (Table 1), No. 23, No. 24, No. 25 (Table 2), No. 45, No. 46, No. 48, No. 51 (Table 3). The goal of ITSs for programming problem solving support is to provide students with intelligent help on each step of programming problem solving - from giving a hint to executing the next step for the student. The typical ITSs for programming of the class 3 are No. 40, No. 46, No. 47 (Table 3).

Most of the ITSs for programming have been developed to learn to write programs (class 2 and class 3). ITSs for programming are useful for first year computer science students and non-major students. When using these systems, most students can remove compilation errors quickly because the error messages generated by the ITSs for programming are usually very informative. As a consequence, the amount of time students spend on each programming exercise is reduced substantially. Besides, a large number of the students remove the error on their own and this helps in reducing the number of questions asked to the teachers [4].

The following section presents a brief review of the current studies related to this work.

Table 1. *ITSs for programming from 1976 to 1999.*

No.	System name	Authors	Year	Programming Language	Programming Paradigm
1	The computer as a tutorial laboratory: The Stanford BIP project.	Barr,A.,Beard,M.,&Atkinson,R.C.	1976	BASIC	Imperative
2	Meno-ii: An intelligent tutoring system for novice programmers	Soloway,E.M.,Woolf,B.,Rubin,E.,&Barth,P.	1981	Pascal	Imperative

3	Design consideration so fan intelligent tutoring system for programming languages	Elsom-Cook,M.	1984	Lisp	Function
4	The LIS Ptutor	Anderson,J.R.,&Reiser, B.J.	1985	Lisp	Function
5	PROUST: Knowledge-based program understanding	Johnson,W.L.,&Soloway ,E.	1985	Pascal	Imperative
6	Talus: Automatic Program Debugging for Intelligent Tutoring Systems	Murray,W.R	1986	Lisp	Function
7	Toward san intelligent tutoring system for Pascal programming	Doukidis,G.I.,Angelides, M.C.,&Harlow,J.L.	1988	Pascal	Imperative
8	Bridge: Intelligent tutoring with inter mediate representations	Bonar,J.G.,&Cunningha m,R.	1988	Pascal	Imperative
9	ItsAda: An Intelligent Tutoring System for the ADA Programming Language	DeLooze,L.L.	1991	Ada	Imperative
10	An integrated knowledge-based intelligent programming environment for novice programmers	Ueno,H.	1991	Pascal	Imperative
11	Automatic debugging of Prolog programs in a Prolog intelligent tutoring system	Looi,C.K.	1991	Prolog	Logic
12	Hyperex: An intelligent tutoring hypertext system for learning programming	Altamura,O.,&Roselli,T.	1995	Pascal	Imperative
13	ELM-ART: An intelligent tutoring system on World Wide Web	Brusilovsky,P.,Schwarz, E.,&Weber,G.	1996	Lisp	Function
14	An intelligent tutoring system for introductory C language course	Song,J.S.,Hahn,S.H.,Tak ,K.Y.,&Kim,J.H.	1997	C	Imperative
15	Extraction of problem description from sample program for knowledge-based programming tutoring	Hahn,S.H.	1997	C	Imperative
16	Automatic diagnosis of student programs in programming learning environments	Xu,S.,&Chee,Y.S.	1999	Smalltalk	Object Oriented

Table 2. *ITSs for programming from 2000 to 2010*

No.	System name	Authors	Year	Programming Language	Programming Paradigm
17	Model-based reasoning for domain modeling in a web-based intelligent tutoring system to help students learn to debug C++programs	Kumar, A.N.	2002	C++	Imperative, Object Oriented
18	Transformation-based diagnosis of student programs for programming tutoring systems	Xu,S.,&Chee,Y.S.	2003	Smalltalk	Object Oriented

19	Propat: A programming ITS based on pedagogical patterns. In Intelligent Tutoring Systems	Delgado,K.V.,&deBarros,L.N	2004	C	Imperative
20	Intelligent tutoring and knowledge base creation for the subject of computer programming	Muansuwan,N.,Sirinaovakul,B.,&Thepruangchai,P.	2004	C	Imperative
21	Exercise sequence adaptation in programming education	Taguchi,H.,&Shimakawa,H	2004	C	Imperative
22	A web-based intelligent tutoring system for computer programming	Butz,C.J.,Hua,S.,&Maguire,R.B.	2004	C++	Imperative, Object Oriented
23	Haskell-Tutor: An Intelligent Tutoring System for Haskell Programming language	Xu,L.,&Sarrafzadeh,A.	2004	Haskell	Function
24	A Dialogue-Based Tutoring System for Beginning Programming	Lane,H.C.,&VanLehn,K.	2004	Pascal	Imperative
25	Guided programming and automated error analysis in an intelligent Prolog tutor	Hong,J.	2004	Prolog	Logic
26	Teaching the tacit knowledge of programming to novices with natural language tutoring	Lane,H.C.,&VanLehn,K	2005	C,Java	Imperative, Object Oriented
27	The Intelligent Web-Based Tutoring System using the C++ Standard Template Library	Lee,C.,&Baba,M.S.	2005	C++	Imperative, Object Oriented
28	Proto type Model of Tutoring System for Programming	Dadic,T.,Stankov,S.,&Rosic,M	2006	BASIC	Imperative
29	A multi-agent intelligent tutoring system for learning computer programming	Sierra,E.,Hossian,A.,Britos,P.,Rodriguez,D.,&Garcia-Martinez,R.	2007	C++,Java	Imperative, Object Oriented
30	Developing an intelligent tutoring system for students learning to programin C++	Naser,S.S.A.	2008	C++	Imperative, Object Oriented
31	M-PLAT: Multi-Programming Language Adaptive Tutor	Nuez,A.,Fernández,J.,Garcia,J.D.,Prada,L.,&Carretero,J.	2008	Java	Object Oriented
32	J-LATTE: a Constraint-based Tutor for Java.	Holland,J.,Mitrovic,A.,&Martin,B.	2009	Java	Object Oriented
33	An intelligent tutoring system for C++	Mishra,K.,&Mishra,R.B.	2010	C++	Imperative, Object Oriented
34	Algo Tutor: from algorithm design to coding	Yoo,S.,&Yoo,J.	2010	C++	Imperative, Object Oriented
35	Design, Development and Evaluation of the Java Intelligent Tutoring System	Sykes,E.R.	2010	Java	Object Oriented

Table 3. *ITSs for programming from 2011 to 10/15/2015*

No.	System name	Authors	Year	Programming Language	Programming Paradigm
36	Research and application of plan recognition in Intelligent Tutoring System	Liu,L.,Wang,H.,Li,C.,& Zhao,C.	2011	Java	Object Oriented
37	Using weighted constraints to build a tutoring system for logic programming	Le,N.T.	2011	Prolog	Logic
38	An Intelligent E-Learning System for Beginner Programming-Using Analogical Reminder for Error Classification and Explanation	Pollack,R.	2011	Scheme	Imperative, Object Oriented, Function
39	Program representation for automatic hint generation for adata-driven novice programming tutor.	Jin,W.,Barnes,T.,Stamper,J.,Eagle,M.J.,Johnson, M.W.,&Lehmann,L.	2012	C++	Imperative, Object Oriented
40	ASK-ELLE: aHaskell Tutor	Gerdes,A.	2012	Haskell	Function
41	Improving testing abilities of a programming tutoring system	Vesin,B.,Klasnja-Milicevic,A.,&Ivanovic, M.	2013	Java	Object Oriented
42	Intelligent tutoring system for learning PHP	Weragama,D.S.	2013	PHP	Scripting/ Dynamic
43	Automated feedback generation for introductory programming assignments	Singh,R.,Gulwani,S.,&Solar-Lezama,A.	2013	Python	Scripting/ Dynamic
44	KEMCs: A set of student's characteristics for modeling in adaptive programming tutoring systems	Chrysafiadi,K.,&Virvou, M	2014	C	Imperative
45	Incorporating anchored learning in a C# intelligent tutoring system	Hartanto,B.	2014	C#	Object Oriented
46	Strategy-based feedback for imperative programming exercises	Keuning,H.	2014	Java,PHP	Object Oriented, Scripting
47	Data-Driven Program Synthesis for Hint Generation in Programming Tutors. In Intelligent Tutoring Systems	Lazar,T.,&Bratko,I.	2014	Prolog	Logic
48	Introducing Code Adviser: ADFA-driven Electronic Programming Tutor	Ade-Ibijola,A.,Ewert,S.,&Sanders,I.	2015	C++	Imperative, Object Oriented
49	An exploration of data-driven hint generation in an open-ended programming problem	Price,T.W.,&Barnes,T.	2015	Grace	Object Oriented
50	Adaptive structure metrics for automated feedback provision in Java programming	Paaßen,B.,Mokbel,B.,& Hammer,B.	2015	Java	Object Oriented
51	Learning to program using hierarchical model-based debugging	deBarros,L.N.,Pinheiro, W.R.,& Delgado,K.V.	2015	Java	Object Oriented

III. RECENT APPROACHES

1. The Genetic Programming (GP) Approach

A genetic programming algorithm for evolving imperative programs using memory, selection and iterative programming constructs was implemented. The GP approach was able to evolve solution programs for 10 programming problems taken from a first year course on programming [5]. One major drawback of this approach is the ability to evolve solution programs for only 10 programming exercises.

2. The Deterministic Finite Automaton (DFA) Approach

This approach comprised the following steps:

1. Take a model program for a programming problem written in C++ (provided by teacher/expert because they are experts in their field and their solutions serve as examples for students) with a number of test cases, cleans up and granulates the model program,
2. Generate all possible variations of the model program,
3. Construct a DFA from the solution space, taking each solution as a string,
4. Attempt to compare a buggy student program to the finite list of program strings accepted by the problem's DFA, and
5. Depending on the student's plan and output correctness, report the discovered bugs with suggested repairs or declare the student's program as correct [6].

Code Adviser [6] is an ITS for programming C++. This system uses the approach of DFA to understand and find semantic bugs in student programs written in C++ programming language.

However, as noted by the authors, this approach is not robust, it is a proof of concept that we have used to demonstrate how a tool can be used to tutor student programmers based on DFAs of alternative solutions and bug detection algorithms [6].

3. The Constraint Based Modeling (CBM) Approach

This approach uses constraints to model a space of correct solutions rather than enumerating them. A constraint represents a domain principle or specifies a property of correct solutions. A set of constraints divides the space of solutions into two subspaces: the inner space for correct and the outer space for incorrect solutions as Figure 1. illustrates. Whenever a solution violates a constraint, that solution falls into the outer space, and a CBM tutoring system derives a feedback associated to that violated constraint [7-8]. Using this approach, Holland [7] proposed an ITS for programming Java named J-LATTE (Java Language Acquisition Tile Tutoring Environment). Le [8] also used the CBM approach in his PhD thesis to develop a system (INCOM) which is an ITS for Prolog programming. However, a programming exercise with this approach fails to take the imperative programming languages into account.

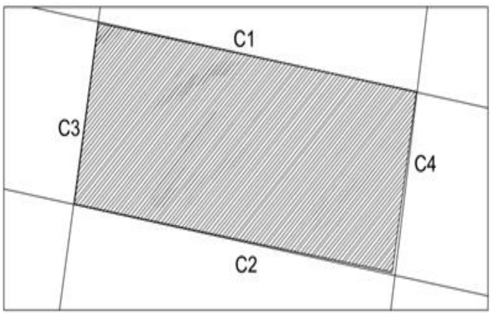

Figure 1. *A solution space determined by the constraints.*

(Ci: constraint, ▨: space of correct solutions, ☐ *: space of incorrect solutions)*

4. The Program Transformation Based Approach

This approach is used extensively to identify alternate solutions to a given programming exercise to convert the program code into a standardized form. The standardized form is then compared against a solution that is stored in the same standardized form. Different standardized forms have been proposed.

A linkage graph is a directed acrylic graph whose nodes represent program statements and directed edges indicate dependencies between the different statements. This graph is represented as a two dimensional matrix. Equivalent programs have equal matrices, thereby allowing accepting alternative solutions to a single exercise [9]. However, the published work only deals with the assignment statement. The probability that this method will be able to produce equal matrices for logically equivalent programs using other programming structures is yet doubtful. The completed student solution is converted into an AST, which captures the structure of a program. The form of the AST is dependent on the structure of the program. Therefore, alternative solutions to a single program have different ASTs. This means that the AST obtained from the student's program is converted to a standard form using a set of rules. Once a student's solution has been converted into an AST, the systems can gather relevant information on what data structures and algorithms the student is used by checking the tree [10-11]. Jin and his colleague [9] used this approach to build an ITS for Python programming. They used linkage graphs to represent correct student solutions. Although this method seems suitable for identifying alternative solutions to small programming exercises, it is difficult to be sure that it is expandable for larger programs.

5. The Data-Driven Approach

The program synthesis method models programming directly in terms of textual edits allow us to trace student actions more closely. This one is generative: given an incorrect program, it finds a sequence of edits that transforms it into a correct solution. The goal of this method is to synthesize new programs from an incorrect solution [12]. One major drawback of this approach is the search algorithm. When searching for edit sequences, the scoring function only considers edits in the current sequence. According to the machine learning method, solution spaces are automatically clustered by machine learning techniques operating on the sets of student solutions. Based on these structured solution spaces, the system proposes feedback provision strategies that employ example based learning methods comparing student solution attempts to appropriate sample solutions [13]. The key problem with this method is that, let us assume that a set of correct student solutions is given for a programming exercise, feedback can then be generated based on a sufficient number of examples, which are essentially high quality solutions, either included in a database of student solutions or provided by teachers/experts as designated sample solutions. The ITAP (Intelligent Teaching Assistant Programming) combines algorithms for state abstraction, path construction and state reification to fully automate the process of hint generation, even when given states that have not occurred in the data before. ITAP makes it possible to generate a full chain of hints from any new code state to the closest goal state. Further, the ITAP is an instance of a self-improving ITS, a tutor that continually improves its ability to provide hints that are personalized to each students' individual solution to a programming exercise. The ITAP requires a two pieces of expert knowledge to run independently, though this knowledge is kept to a minimum. The needed data are: At least one reference solution to the problem (e.g. a teacher/expert exemplar) and a test method that can automatically score code (e.g. pairs of expected input and output). Both reference solutions and test methods are already commonly created by teachers/experts in the process of preparing assignments, so the burden of knowledge generation is not too large [10]. A major limitation of this method is that it relies on the existence of test cases that can measure the correctness of solutions. Though there are a large number of programming exercises which can easily be tested using input/output sets, there are many other programming exercises which are difficult to test; for example, graphical assignments, interactive programs and programs using randomization. Piech et al. [14] have provided a definition of Problem Solving Policy (PSP): "PSP is a decision for any partial solution as to what next partial solution a student should take" (see the example in Figure 2). Furthermore, they claim that "data of how previous students navigated their way to the final answer can be leveraged to autonomously understand the landscape of such assessments and enable hints for future students," thereby potentially increasing future student retention. Piech et al. has offered a comparative analysis on tested multiple solution space generation algorithms (including other's path constructions and their problem solving policies: all ten algorithms) from a MOOC (Massive Open Online Course: Code.org) to determine how often the selected next states matched the next states chosen by teachers/experts. They found that several of the algorithms had a high match rate, indicating that this new approach has great potential to generate the hints that students will benefit the most from.

6. The Variation Theory Based Method

Glassman et al. [15] proposed the use of variation theory to explain differences in student submissions. Applying variation theory to programming exercises essentially means using a hierarchical approach: first, distinguishing clusters of approaches to solving a problem and then within each cluster identifying differences in various students' implementation of a particular approach. This "divide and conquer" method has many other use cases, including pairing students based on their problem-solving strategies and making it easier to illustrate to students the relative merits of different approaches. Gulwani et al. [16] studied a large number of functionally correct student solutions to introductory programming assignments and observed: (1) There are different algorithmic strategies, with varying levels of efficiency, for solving a given problem. These different strategies merit different feedback. (2) The same algorithmic strategy can be implemented in countless different ways. However, the key problem with this method is that a teacher has to define an algorithmic strategy.

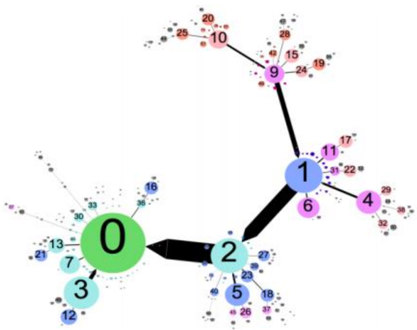

Figure 2. *Each node is a unique partial solution, node 0 is the correct answer. The edges show what next partial solution they think a teacher/expert would suggest students move towards.*

IV. RELATED WORK

Many of these used approaches traditional to ITS design, such as providing examples, simulation or scaffolding dialogue are mentioned in a review of such systems provided by Le and colleagues [17]. However, the authors do not yet consider recent data-driven approaches, which are mostly feedback-based systems, for example ITAP [10]. A recent survey by Hooshyar [18] summarizes some of this work. However, this research does not discuss about approaches for covering solution space for ITSs for programming.

V. CONCLUSIONS

The objective of this research is to identify and categorize the current approaches to handle large solution space of ITSs for programming.

The primary contributions of this paper are:

- A review of existing ITSs for programming with many different programming languages and different programming paradigms from year 1976 to 2015, and
- A brief review of recent approaches for solution space problem in the context of ITSs for programming.

In the context of student solutions, existing ITSs for programming may be divided into two categories which are 1) incomplete/partial solutions and 2) complete/final/full solutions. Data-driven approach is suitable for ITSs for programming with incomplete/partial solutions. Data-driven methods have been a successful approach to covering solution spaces for programming exercises of ITSs for programming. An advantage of this data-driven approach is that the instructor does not have to provide any inputs. On the other hand, it requires the existence of a large data set. Furthermore, in this approach, most of these algorithms have only been evaluated on the collected student programming exercise solving traces and the ones that are being tested on real students are implemented in online learning environments such as MOOCs instead of in individual classrooms. In a typical course, each topic is presented in a specific lecture on a specific date according to the course curriculum and, at each moment of time, students are expected to focus on the current topic(s). To reflect this classroom practice, Data-Driven ITSs for programming should maintain the course schedule, which associates each topic with the date of its presentation.

Only one system, the ITAP is an instance of a self-improving ITS, a tutor that continually improves its ability to provide hints that are personalized to each students' individual solution to a programming exercise. It does so by updating the solution space every time a student attempts a solution and recalculating optimal paths.

In summary, for the informants, in this study, compared to others, the data-driven approach is flexible.

Currently, most of these systems construct the domain models manually. They take much time to construct, especially for exercises with very large solution spaces. One of the major challenges associated with handling ITSs for programming comes from the diversity of possible code solutions that a student can write. The use of data-driven approaches to develop these ITSs is just starting to be explored in the field. Given that this is still a relatively new research field, many challenges are still remained unsolved.

REFERENCES

[1] Keuning, H. (2014). Strategy-based feedback for imperative programming exercises. PhD thesis, Utrecht University.

[2] Glassman, E. L., Scott, J., Singh, R., Guo, P. J., & Miller, R. C. (2015). OverCode: Visualizing variation in student solutions to programming problems at scale. ACM Transactions on Computer-Human Interaction (TOCHI), 22(2), 7.

[3] Weragama, D. S. (2013). Intelligent tutoring system for learning PHP. PhD thesis, Queensland University of Technology

[4] Gómez-Albarrán, M. (2005). The Teaching and Learning of Programming: A Survey of Supporting Software Tools. *The Computer Journal*, *48*(2), 130-144.

[5] Igwe, K., Pillay, N., Corchado, E., Yun-Huoy, C., & Ma, K. (2013). Automatic Programming Using Genetic Programming. In Proceedings of the World Congress on Information and Communication Technologies., Hanoi, Vietnam (pp. 339-344).

[6] Ade-Ibijola, A., Ewert, S., & Sanders, I. (2015). Introducing Code Adviser: A DFA-driven Electronic Programming Tutor. In Implementation and Application of Automata (pp. 307-312). Springer International Publishing.

[7] Holland, J. (2009). A Constraint-Based ITS for the Java Programming Language (Doctoral dissertation, University of Canterbury). Retrieved from https://www.csse.canterbury.ac.nz/research/reports.

[8] Le, N.-T. (2011). Using weighted constraints to build a tutoring system for logic programming. (PhD thesis), University of Hamburg, Germany.

[9] Jin, W., Barnes, T., Stamper, J., Eagle, M. J., Johnson, M. W., & Lehmann, L. (2012, January). Program representation for automatic hint generation for a data-driven novice programming tutor. In Intelligent Tutoring Systems (pp. 304-309). Springer Berlin Heidelberg.

[10] Rivers, K., &Koedinger, K. R. (2015). Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. International Journal of Artificial Intelligence in Education, 1-28.

[11] Truong, N. (2007). A web-based programming environment for novice programmers (Doctoral dissertation, Queensland University of Technology).

[12] Lazar, T., &Bratko, I. (2014, January). Data-Driven Program Synthesis for Hint Generation in Programming Tutors. In Intelligent Tutoring Systems (pp. 306-311). Springer International Publishing.

[13] Gross, S., Mokbel, B., Paassen, B., Hammer, B., &Pinkwart, N. (2014). Example-based feedback provision using structured solution spaces. International Journal of Learning Technology 10, 9(3), 248-280.

[14] Piech, C., Sahami, M., Huang, J., &Guibas, L. (2015, March). Autonomously Generating Hints by Inferring Problem Solving Policies. In Proceedings of the Second (2015) ACM Conference on Learning@ Scale (pp. 195-204). ACM.

[15] Glassman, E. L., Singh, R., & Miller, R. C. (2014). Feature engineering for clustering student solutions. In Proceedings of the first ACM conference on Learning@ scale conference (pp. 171-172). ACM.

[16] Gulwani, S., Radiček, I., &Zuleger, F. (2014). Feedback generation for performance problems in introductory programming assignments. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 41-51). ACM.

[17] Le, N. T., Strickroth, S., Gross, S., &Pinkwart, N. (2013). A review of AI-supported tutoring approaches for learning programming. In Advanced Computational Methods for Knowledge Engineering (pp. 267-279). Springer International Publishing.

[18] Hooshyar, Danial (2016). A Flowchart-Based Intelligent Tutoring System Model to Improve Students' Problem-Solving Skills. PhD thesis, University of MalayaKuala Lumpur, Malaysia. Retrieved from http://studentsrepo.um.edu.my/6244/1/Thesis.pdf

Corresponding Author:
Nguyen Van Y
Ho Chi Minh City University of Technology and Education
Email: nguyenvany_tcb@yahoo.com