Generating Data-Driven Hints for Programming Exercises in Code-Writing Intelligent Tutoring Systems

Y Nguyen Van[1], Hieu Bui Trong[2]

[1]Manager of Examination and Quality Assurance Office, Ho Chi Minh City Cadre Academy, Ho Chi Minh City, Vietnam

[2] Faculty of Information Technology, Ho Chi Minh City University of Transport, Ho Chi Minh City, Vietnam

Abstract

The skill to write code accurately and fluently is a core competency for every engineering student. In any engineering discipline in universities, introductory programming is an essential part of the curriculum. However, for many novice 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.

Currently, mostly these systems manually construct the domain models. 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. This paper presents a review of analysis techniques that are requested to generate data-driven hints in code-writing ITSs.

Keywords: *Programming Exercises, Intelligent Tutoring Systems, Data-Driven Hint Generation, Programming Tutors.*

1. Introduction

Programming skills are becoming a core competency for almost every profession and thus, computer science education is being integrated in the curriculum for almost every study subject [1]. However, many students find great difficulty with the learning of programming and it becomes a barrier to their further studies of computer science and other disciplines. This difficulty is in large part due to students' inabilities to solve their programming exercises, and this may discourage them to progress further when help can be obtained immediately. In order to address this problem, various approaches have been proposed to help students learn solving programming exercises. Traditionally, face-to-face and one-to-one human tutoring had been the best option for tutor. However, human tutors are not always available and that's why computer based tutoring is developed to provide as an alternative support. Intelligent Tutoring System (ITS) is an example of computer-based tutoring which is developed emulating the human tutor [2]. According to VanLehn [3], an Intelligent Tutoring System that is designed with the ability to understand the coding to a low level of granularity in its advice can be just as effective as human tutor. ITSs for programming are useful particularly for first year computer science students and non-major students [4].

A current trend in the ITSs for programming world is to use data-driven techniques to give hints to users of ITSs for programming [5, 6, 7, 8, 9, 10, 11, 1, 12, 13, 14].

Instead of taking much time for modeling domain knowledge, the data-driven approach uses a mass of correct student programs. The data-driven approach uses correct student solutions in order to construct a solution space that contains all solution states students have created in the past (e.g., in the former semesters of a programming course). The solution states build many possible paths to correct solutions [1].

2. Background

2.1 Intelligent Tutoring Systems

As we stated above, face-to-face and one-to-one human tutoring is the best tutoring field. However, it is extremely expensive in terms of both physical and human resources. ITSs are a natural solution that can be used to address this problem, as they are developed to give personalized feedback and help to students who are working on problems.

The fact the ITSs are formed by three fields: Computer Science, Psychology, and Education, as illustrated in Figure 1, in which, (i) Artificial Intelligence (AI) addresses how to reason about intelligence and thus learning, (ii) Psychology (Cognitive Science) addresses how people think and learn, and (iii) Education focuses on how to best support teaching/learning [15].

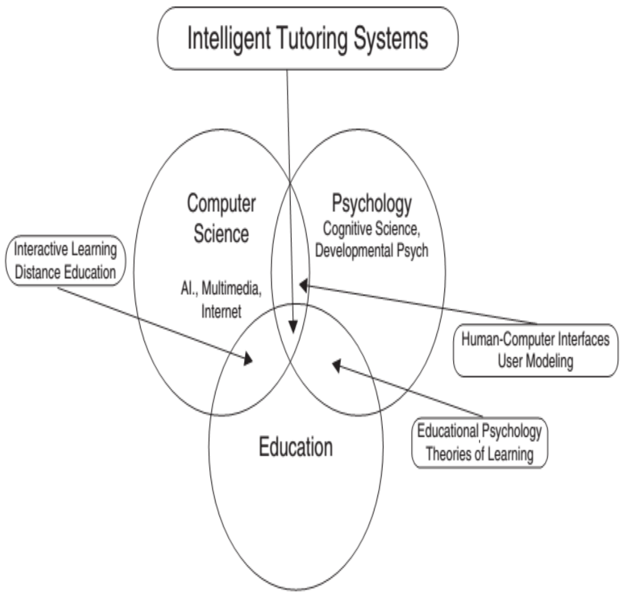

Figure 1 The development of an Intelligent Tutoring System using methods and instruments from three different domains.

According to Lee & Chen [16], an Intelligent Tutoring System (ITS) is a computer system that provides immediate and customized instruction or feedback to learners. The classical architecture of an Intelligent Tutoring System includes the following four components (Figure 2) [17, 18, 19, 20].

- A knowledge domain model that stores the learning content that is taught to students.
- A student model that stores information about the student's knowledge level, abilities, preferences and needs.
- A tutoring (pedagogical) model, which makes student diagnosis and controls the tutoring process and make appropriate instructional decisions based on the information provided by the other components of the ITS.
- A User Interface that allows the system to interact with the user-learner

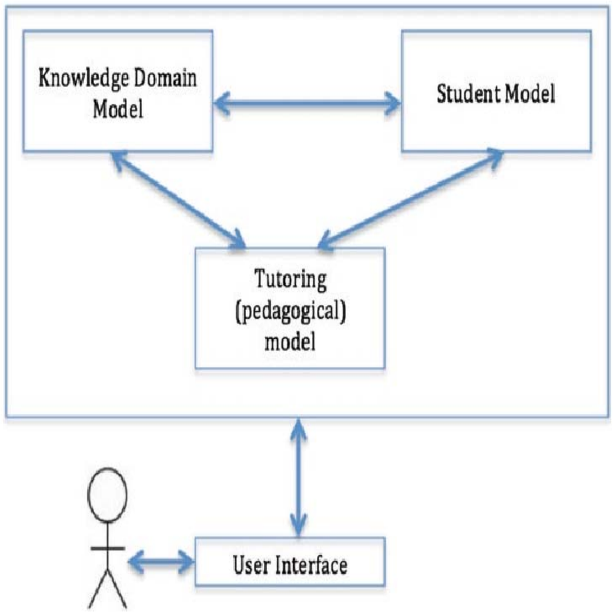

Figure 2 The typical architecture of an ITS.

This traditional view of ITSs is still very accepted by the ITS community. However, recent studies stress functionality over structure [21, 22, 17, 7, 23], describing ITSs as having two main loops [21]: 1) the inner loop and 2) the outer loop (Figure 3) [17]. The inner loop is responsible for providing personalized feedback, hints, and direct problem solving assistance to students. The inner loop also assesses students' competence and registers it on the student model. Using the information that is obtained about the student, the outer loop performs task selection.

This work is inspired from VanLehn's two loop characterization of tutoring systems. The main task of the outer loop is to select an appropriate programming exercise for the student. The inner loop is responsible for giving hints on student steps.

```
until tutoring is complete, repeat
{
   tutor selects a task;
  until task is complete, repeat
   {
    student executes a step;
    tutor may present a hint;
    tutor updates the student model;
    tutor presents feedback on the step;
                              }
}
```

Figure 3 ITS Loops.

Here, we focus on the inner loop. We do not support an outer loop which can create an overall student model and intelligently choose which programming exercises to show to the student.
Nesbit et al. [24], in their paper, "Work in Progress: Intelligent Tutoring Systems in Computer Science and Software Engineering Education", research on ITSs has accelerated over the last decade, and scholarly interest in such systems has never been greater. ITS have been developed for a wide range of subject domains (e.g., mathematics, physics, biology, medicine, reading, languages, philosophy, information technology and computer science) and for students in primary, secondary and postsecondary levels of education.
Founded on several decades of research on human cognition and intelligence, Intelligent Tutoring System is now a fast growing area in academia and industry. We now turn our attention to some cutting-edge research on Intelligent Tutoring System in a specific learning domain: programming [25].

2.2 Intelligent Tutoring Systems in the programming domain

In the past four decades, a variety of ITSs for programming have been built to provide tutoring services for programming problems. When it comes to functionalities, in general, ITSs for programming can be classified into five types: 1) curriculum sequencing, which constructs for each student an individual learning path, including individual selection of topics to learn, examples, and exercises; 2) intelligent analysis of student's solutions, which focuses more on debugging and error diagnosis for complete student's program; 3) program debugging support, which helps students learn to analyze programs; 4) interactive code-writing problem solving support, which provides students with personalized assistance in each code-writing problem solving step and 5) example based code-writing problem solving support which suggests the most relevant cases or examples to students.
In the context of ITSs for programming, for brevity, we will use the term "ITSs for code-writing" to describe to the ITSs for programming for interactive code-writing problem solving support.

2.3 Generating Automatic Hints in Code-Writing ITSs

Several recent studies deal with the problem of helping students to learn programming, in particular by giving them useful hints in real time while they are coding. According to Keuning et al. [26], ITSs for code-writing that focus on the process of solving an exercise are still rare or have limitations: some targeted for declarative programming [27, 6], which is less flexible because they do not support exercises that can be solved by multiple algorithms [28, 29], or only support a static, pre-defined process [30]. Furthermore, it often requires substantial work to add new exercises [31] and tutors can be difficult to adapt by a teacher. ASK-ELLE [11] is an Intelligent Tutoring System for code-writing for learning the higher-order, strongly-typed functional programming language Haskell. They model alternative solution strategies in the system ASK-ELLE through several model programs (e.g. model solutions). This system supports the stepwise development of Haskell programs by verifying the correctness of incomplete programs, and by providing hints. Programming exercises are added to ASK-ELLE by providing a task description for the exercise, one or more model solutions, and properties that a solution should satisfy. The properties and model solutions can be annotated with feedback messages, and the amount of flexibility that is allowed in student solutions can be adjusted. The disadvantage of this strategy-based approach is that their tutor based on model solutions provided by instructors/teachers, because they are experts in their field and their solutions serve as examples for students.

However, variations to these model solutions are boundless. Programming exercises are characterized by huge and expanding solution spaces, which cannot be covered by manually designed hints. According to Irfan and Gudivada [25], this is a vastly challenging problem, mainly because even for very simple programming tasks there are a multitude of different solution approaches, both syntactically and semantically. Even if we restrict the semantic aspect (i.e., the underlying algorithm) to a single one, the syntactic variations of implementing the algorithm present a daunting task for hint generation.

For such exercises, ITSs for code-writing are still possible to collect implicit data in terms of solutions given by students or teachers/experts. The data-driven approach is particularly useful when it is hard to come up with a more or less complete set of model solutions. According to Rivers and Koedinger [7], data-driven Intelligent Tutoring System is a subfield of ITS where decision-making is based on the previous student's work instead of a knowledge base built by experts or an author-mapped graph of all possible paths.

3. Reviewing of Data-Driven Hint Generation Techniques for Code-Writing ITSs

New research efforts to tackle broader programming exercises are at a nascent stage and use previous students' solutions to a programming exercise to generate hints for a new student who is working on the same exercise. Recently, there are two types of data-driven hint generation in ITSs for code-writing: hint generation has focused on code correctness and hint generation for code style. In this research work, we focus on the hint generation for code correctness.

3.1 Program Synthesis Techniques

In his doctoral dissertation, Singh [31], used error models and program sketches to find a mapping from student current programs to a model solution. Rather than relying on a predefined set of solutions, he used program synthesis to generate a new solution from the student's current program. However, according to Terman [32], this system requires experts/teachers to define an error model specific to each programming exercise, and only supports a subset of Python.
Lazar and Bratko [6] has relied on analyzing the single-line edits made by students between submissions, and then using those edits to attempt to find a correct solution for the Prolog program. Those edits could then be used as a source for hints to be supplied to the new student. However, their technique requires a set of test cases to evaluate generated programs [12].
Perelman et al. [33] published their study to use all common expressions that occurred in students' code to create a database of source code that was then used for hint generation.
According to Rivers and Koedinger [7], these techniques have great potential for supporting new and obscure solutions, but also have the drawback of only working on solutions which are already close to correct; they all tend to fail when the code has many different errors.

3.2 Cluster Based Techniques

Gross and colleagues [34] used clustering to infer clusters of computer programs and select the most similar sample solution for hint generation. When the student requires a hint on how to change her/his code to get closer to a correct solution, it can be compared to a similar example from the cluster, and the dissimilarities between her/his code and the example code can be contrasted or highlighted in order to help the student to improve her/his own solution. As noted by the authors, the challenge with this approach is the derivation of solution steps from sample complete solutions in order to reduce the effort for modeling examples.
Paaßen et al. [13] introduced an alternative representation of computer programs for classification and error detection in ITSs, namely execution traces. The trace representation can be applied to identify erroneous programs, enabling an Intelligent Tutoring System to detect whether a student has finished a task or still needs to continue. However, they concluded that a syntactic representation is necessary when a program does not yet compile or crashes and wherever the high level of abstraction applied by a program trace is not helpful (e.g. when teaching certain syntactic constructs).

3.3 Recommendation Technique

Zimmerman and Rupakheti [35] represent a framework that can help students in their coding process by recommending specific code edits relevant to their codes. They use a pq-Gram tree edit distance algorithm to match a student's program to its closest counterpart in a database of correct solutions, as well as to identify the set of insertions, deletions and relabeling that will directly transform the student's AST into this solution. According to the authors, the disadvantages of this method involve the following three aspects: AST based program analysis, semantic similarity of programs and usability testing.

3.4 Case Based Reasoning Technique

In the newest paper by Freeman et al. [36], they use a case-based reasoning (CBR) approach, which they call Abstract Syntax Tree Retrieval (ASTR) to data mine prior solutions contained in a large dataset. This system requires no prior knowledge of the problem being solved. It uses CBR and the grammar of the programming language to retrieve a prior solution with high similarity to a struggling student's failing submission. The results achieved by their system are encouraging. However, as noted by the authors, the system contains no information about the programming problem prior to observing successful submissions. Additionally, their system has no understanding of Python syntax.

3.5 Hint Factory Based Techniques

In general, the basic technique in this new line of work is to first represent the previous student–tutor interactions in the form of a graph. When a new student asks for a hint, that student's interaction In the newest paper by Freeman et al. [36], they use a case-based reasoning (CBR) approach, which they call Abstract Syntax Tree Retrieval (ASTR) to data mine prior solutions contained in a large dataset. This system requires no prior knowledge of the problem being solved. It uses CBR and the grammar of the programming language to retrieve a prior solution with high similarity to a struggling student's failing submission. The results achieved by their system are encouraging. However, as noted by the authors, the system contains no information about the programming problem prior to observing successful submissions. Additionally, their system has no understanding of Python syntax.
pattern is matched with some part of the graph and the student is directed to an appropriate next step that ultimately leads to a solution. It is not hard to imagine the potential impact of such work on any Intelligent Tutoring System that teaches programming [25].
Barnes and Stamper [37] designed the Hint Factory to use student problem-solving data for automatic hint generation in a propositional logic tutor. This approach uses student data to build a Markov decision process of student problem-solving strategies to serve as a domain model to automatic hint generation. The Hint Factory operates on a representation of a problem called a directed graph where each node represents a student's state at some point in the problem solving process, and each edge represents a student's action that alters that state. A solution is represented as a path from the initial state to a goal state. A student requesting a hint is matched to a previously observed state and directed on a path to a goal state. The Hint Factory approach has been extended to work in other domains more closely related to programming.
Fossati et al. [38, 39] implemented Hint Factory in the iList tutor that helps students learn linked list, a demanding topic in information technology and computer science education. Fossati et al. [39] also concluded that their tutor produced equivalent learning gains to a human tutor.
Using the Hint Factory approach, Jin et al. [5] use linkage graphs to represent program states. A linkage graph is an acyclic graph consisting of nodes representing code statements and directed edges representing the order of the statements determined by which variables are read and assigned to in each statement. However, Keuning [40] points out that multiple existing student solutions should be available with the risk that a specific alternative to solve the exercise might not be recognized. On the other hand, as noted by the authors, the challenge with this method is the determination of strategies for hint presentation.
Rivers and Koedinger [41, 42, 7] propose a data-driven approach to create a solution space consisting of all possible paths from the problem statement to a correct solution. This approach borrows heavily from the Hint Factory, but also extends it by enhancing the solution space, creating new edges for states that are disconnected instead of relying on student-generated paths. ITAP (Intelligent Teaching Assistant for Programming) [7] makes it possible to generate hints for never-seen-before states, which the original Hint Factory could not do. ITAP combines algorithms for state abstraction (the process of reducing syntactic variability in code states), path construction (determining which steps a student should take to improve their solution), and state reification (re-individualizing the resulting edits into personalized hint messages) to fully automate the process of hint generation. However, the disadvantages of the state abstraction are that it is difficult to see if it is expandable for larger programs [18], it is limited by the fact that a set of semantic-preserving AST

(Abstract Syntax Tree) transformations, the task of designing transformation rules is highly nontrivial, however, and differs for each programming language [43], the resulting state space is much more sparse and resistant to canonicalization [9]. On the other hand, as noted by the authors, the path construction algorithm could be modified to further improve the performance. Furthermore, a recent comparative analysis by Piech et al. [10], tested multiple solution space generation algorithms (including Rivers' and Koedinger's path construction algorithm and nine other algorithms) to determine how often the selected next states matched the next states chosen by experts/teachers. All of the ten algorithms that were running and generating next states were tested against the gold standard data they had collected. They found that several of the algorithms had a high match rate (above 95% for problem P_A and above 83% for problem P_B), the Rivers' and Koedinger's path construction algorithm had a lower match rate (72.9% for P_A and 78.2% for P_B). Price and Barnes [8] improve this approach by proposing a novel subtree-based state matching technique that will find partially overlapping solutions to generate feedback across diverse student programs. Rather than finding an entire state which matches the student, they look for any states with a matching subtree and find advice that applies within that subtree. Literature has not indicated that this algorithm has been implemented and tested.

Piech et al. [10] use path construction to interpolate between two consecutive states on a solution path which differ by more than one edit. This is useful to smooth data when student code is recorded in snapshots that are too far apart. In the other hand, they tested multiple solution space generation algorithms to determine how often the selected next states matched the next states chosen by experts/teachers.

Price and Barnes [9] present a novel method for addressing the data sparsity problem by focusing on minimal-distance changes between students. However, according to Price et al. [12], data-driven hints have not been evaluated on broader programming exercises in novice programming environments, and may not be well equipped to handle them.

Price et al. [12] present a new data-driven algorithm, based on the Hint Factory, to generate hints for these broader programming exercises. As noted by the authors, a major limitation of this work is the reliance on a single programming exercise for evaluation.

4. Discussion

The main objective of the data-driven approaches is to minimize time and expert knowledge to create ITSs for code-writing. In addition to the model solutions, the program synthesis approaches also require experts/teachers provide error models consisting of common mistakes that students are making on a given programming exercise (e.g. the approach of Singh [31]) and a set of test cases to evaluate generated programs (e.g. Lazar and Bratko [6]). This manual effort of creating error models and a set of test cases is time-consuming and tedious for instructors. On the other hand, it sometimes even prohibitively expensive to find correction rules for mistakes that occur infrequently in practice. One major drawback of the cluster based approaches is that it uses datasets of solutions that need to be modeled solution steps manually by experts/teachers. The process is time consuming and tedious (e.g. the approach of Gross et al, [34]). The recommendation approach of Zimmerman and Rupakheti [35] also requires experts/teachers create/find different model solutions of the programming exercises.

In terms of expert knowledge, the Hint Factory based approaches are suitable for generating hints in ITSs for code-writing. These approaches only require a two pieces of expert knowledge to run independently, though this knowledge is kept to a minimum. The needed data is: (1) at least one reference solution to the problem (e.g. a model solution) and (2) a test method that can automatically score code (e.g. pairs of expected input and output). Both model solutions and test methods are already commonly created by experts/teachers in the process of preparing programming exercises, so the burden of knowledge generation is not too large.

This study surveys the existing ITSs for code-writing that are solely based on data-driven hint generation to conclude that they differ from each other in at least the following ways:

1. Representation of student's current code
2. Immediate representation of computer programs
3. Extracting distinct solutions of a programming exercise (Preprocessing)

4. Granularity of the code state used. Granularity refers to the smallest level of detail that a program (source code) is divided into
5. Automatically modeling solution steps
6. Programming language

In the context of data-driven ITSs for code-writing, despite the research efforts in recent years, however, generating data-driven hints is still having some problems.

(1) Semantic similarity. At the heart of data-driven ITSs for code-writing is the notion of program similarity. Measuring the similarities and dissimilarities between programs plays a crucial role in data-driven ITSs. Edit distances have been used as a measurement for the similarity of programs. Most existing systems represent programs as abstract syntax trees (ASTs), however, it is known that the tree edit distance problem is NP-hard [44]. Furthermore, according to Zimmerman and Rupakheti [35], one major pitfall of AST representations of source code is the loss of behavioral information. Because syntactically different programs can behave equivalently, the AST-based approach toward finding differences will incorrectly classify many pieces of source code as different even though they accomplish the same task. As shown in the paper by Piech et al. [10], the edit distance metric between such trees are not discriminative enough to be used to share feedback accurately since programs with similar ASTs can behave quite differently. How to extract distinct solutions from a large dataset consisting of learners' solution attempts and a sample solution created by teachers/experts efficiently and precisely is an unresolved problem [45].

(2) Representation of immediate programming steps. Hosseini et al. [46] showed that (1) most of the students tend to incrementally build the program and improve its correctness; (2) intermediate programming steps are important, and need to be taken into account for providing better feedback to students. As discussed in detail in [3], an Intelligent Tutoring System designed with adequate level of granularity in providing advice can be just as effective as support provided by human tutor. The common approach in data-driven ITSs for code-writing is to take periodic snapshots of a student's code and treat these as states, connecting consecutive snapshots in the graph [8, 12, 41, 42, 7, 5, 30, 38, 39]. They captured snapshots every time students compiled or saved their code, but this is not an accurate representation of a unit of work. Different students might exhibit different types of behaviors regarding how often they save or compile their programs [47]. In any case, the chosen granularity plays a large role in the quantity of explorable states [48, 49]. Hovemeyer et al. [50] also concluded that use of fine-grained edit information can be help to overcome the issue of infrequent student submissions. So it is a challenge for future work to study other representations.

(3) Deriving automatically solution steps from complete sample solutions. According to [51], when students are learning to program, it is important that the process of creating a program is shown step by step. From the experiment, Gross and colleagues [34] showed that solutions modeling steps of problem solving are more appropriate for beginners than complete sample solutions.

4. Conclusions

In this paper, we have identified major approaches to generate data-driven hints for ITSs for code-writing. They are: (1) program synthesis, (2) cluster based, (3) recommendation, (4) case based reasoning and (5) hint factory based.

In summary, in this work, the gaps we identified that provide the motivation for future researches are listed below.

1) Representation of the student's current code

In the context of Hint Factory based approaches to generate data-driven hint for ITSs for code-writing, a student's state corresponds to a snapshot of the student's current code. However, as we noted above, the snapshots are captured every time students compiled or saved their code, but this is not an accurate representation of a unit of work.

2) Immediate representation of computer programs

In the most of recent studies, existing data-driven ITSs for code-writing represent program code with ASTs. In the literature, none of the studies present the research on a representation of source code that focuses on graphs, and apply it in the context of generating data-driven hints for ITSs for code-writing.

3) Extracting distinct solutions from the dataset of solutions of a programming exercise

In the context of data-driven ITSs for code-writing in progress, with regard to the Hint Factory based approach, none of the ITSs that extract distinct solutions from the dataset of solutions of a programming exercise. It is big challenge to extract distinct solutions from a large number of correct solutions efficiently and precisely.

4) Modeling automatically solution steps from correct solutions

Clearly, in this literature review, none of the works model automatically solution steps from correct solutions of a programming exercise. How to model automatically solution steps from a large number of correct solutions of a programming exercise is an unresolved problem.

5) Programming language

As we discussed above, in the context of data-driven ITSs for code-writing, it can be seen that although ITSs covering many domains have been developed previously, none of them teach C/C++/Prolog programming.

References

[1] L. Nguyen-Thinh, "Analysis Techniques for Feedback-Based Educational Systems for Programming," In Proceedings of the 4th International Conference on Computer Science, Applied Mathematics and Applications, ICCSAMA 2016, pp. 141-152, 2016.

[2] Chughtai, Rehman, S. Zhang, and ScottyD. Craig, "Usability evaluation of intelligent tutoring system ITS from a usability perspective," In Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 59(1), pp. 367-371, 2015.

[3] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," Journal of Educational Psychologist 46(4), pp. 197-221, 2011.

[4] M. Gómez-Albarrán, "The teaching and learning of programming: a survey of supporting software tools, " The Computer Journal 48(2), p.p.130-144, 2005.

[5] Jin, Wei, T. Barnes, J. Stamper, M. J. Eagle, MatthewW. Johnson, and L. Lehmann, "Program representation for automatic hint generation for a data-driven novice programming tutor," International Conference on Intelligent Tutoring Systems, pp. 304-309, 2012.

[6] L. Timotej and I. Bratko, "Data-driven program synthesis for hint generation in programming tutors," International Conference on Intelligent Tutoring Systems, pp. 306-311, 2014.

[7] R. Kelly and K. R. Koedinger, "Data-driven hint generation in vast solution spaces: a self-improving python programming tutor," International Journal of Artificial Intelligence in Education, p.p.1-28, 2015.

[8] P. Thomas and T. Barnes, "Creating data-driven feedback for novices in goal-driven programming projects," International Conference on Artificial Intelligence in Education, pp. 856-859, 2015.

[9] P. Thomas and T. Barnes, "An Exploration of Data-Driven Hint Generation in an Open-Ended Programming Problem," Educational Data Mining (Workshops), 2015.

[10] P. Chris, M. Sahami, J. Huang and L. Guibas, "Autonomously generating hints by inferring problem solving policies," Proceedings of the Second ACM Conference on Learning@ Scale, pp. 195-204, 2015.

[11] G. Alex, B. Heeren, J. Jeuring and L. Binsbergen, "Ask-Elle: an adaptable programming tutor for Haskell giving automated feedback," International Journal of Artificial Intelligence in Education, pp. 1-36, 2016.

[12] P. Thomas, Y. Dong and T. Barnes, "Generating data-driven hints for open-ended programming," Proceedings of the 9th International Conference on Educational Data Mining, International Educational Data Mining Society, pp. 191-198, 2016.

[13] P. Benjamin, J. Jensen and B. Hammer, "Execution Traces as a Powerful Data Representation for Intelligent Tutoring Systems for Programming," Proceedings of the 9th International Conference on Educational Data Mining, 2016.

[14] F. Paul, I. Watson and P. Denny, "Inferring Student Coding Goals Using Abstract Syntax Trees," International Conference on Case-Based Reasoning, pp. 139-153, 2016.

[15] W. B. Park, Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning, Morgan Kaufmann, 2010.

[16] L. R. Maria and T. T. Chen, "Digital creativity: Research themes and framework," Journal of Computers in Human Behavior 42, pp. 12-19, 2015.

[17] S. G. Soares and J. Jorge,"Interoperable intelligent tutoring systems as open educational resources," Journal of IEEE Transactions on Learning Technologies 6(3), pp. 271-282, 2013.

[18] W. D. Samanthi, "Intelligent tutoring system for learning PHP," eprints.qut.edu.au, 2013. [Online]: Available: http://eprints.qut.edu.au/63202/1/Dinesha%20Samanthi_Weragama_Thesis.pdf. [Accessed: Feb. 12, 2014].

[19] H. Budi, "Incorporating anchored learning in a C# intelligent tutoring system," eprints.qut.edu.au, 2014. [Online]: Available: http://eprints.qut.edu.au/78834/1/Budi_Hartanto_Thesis.pdf. [Accessed: Dec. 12, 2014].

[20] C. Konstantina and M. Virvou, "Student Modeling for Personalized Education: A Review of the Literature," in Advances in Personalized Web-Based Education, pp. 1-24, 2015.

[21] V. Kurt, "The behavior of tutoring systems," International journal of artificial intelligence in education 16(3), pp. 227-265, 2006.

[22] G. Alex, "Ask-Elle: a Haskell Tutor," www.botkes.nl/wp-content/uploads, 2012. [Online]: Available: http://www.botkes.nl/wp-content/uploads/HaskellTutor.pdf. [Accessed: Dec. 20, 2013].

[23] R. Vasile and D. Ştefănescu, "Non-intrusive assessment of learners' prior knowledge in dialogue-based intelligent tutoring systems," Journal of Smart Learning Environments 3(1), 2016.

[24] J. C. Nesbit, Q. L. A. Liu and O. O. Adesope, "Work in Progress: Intelligent Tutoring Systems in Computer Science and Software Engineering Education," Proceeding 122nd Am. Soc. Eng. Education Ann, 2015.

[25] M. T. Irfan and V. N. Gudivada, "Cognitive Computing Applications in Education and Learning," Handbook of Statistics 35, pp. 283-300, 2016.

[26] H. Keuning, B. Heeren and J. Jeuring, "Strategy-based feedback in a programming tutor," Proceedings of the Computer Science Education Research Conference, pp. 43-54, 2014.

[27] J. Hong, "Guided programming and automated error analysis in an intelligent Prolog tutor," International Journal of Human-Computer Studies 61(4), pp. 505-534, 2004.

[28] J. R. Anderson and E. Skwarecki, "The automated tutoring of introductory computer programming," Communications of the ACM 29(9), pp. 842-849, 1986.

[29] P. Miller, J. Pane, G. Meter and S.Vorthmann, "Evolution of novice programming environments: The structure editors of Carnegie Mellon University," Journal of Interactive Learning Environments 4(2), pp. 140-158, 1994.

[30] W. Jin, A. Corbett, W. Lloyd, L. Baumstark and C. Rolka, "Evaluation of guided-planning and assisted-coding with task relevant dynamic hinting," International Conference on Intelligent Tutoring Systems, pp. 318-328, 2014.

[31] R. Singh, "Accessible Programming using Program Synthesis," people.csail.mit.edu, 2014. [Online]: Available:

http://people.csail.mit.edu/rishabh/papers/rishab h_thesis.pdf. [Accessed: Dec. 30, 2015].

[32] S. Terman, "GroverCode: Code Canonicalization and Clustering Applied to Grading," up.csail.mit.edu/other-pubs, 2016. [Online]: Available: http://up.csail.mit.edu/other-pubs/seterman-thesis.pdf. [Accessed: July. 28, 2016].

[33] D. Perelman, S. Gulwani and D. Grossman, "Test-driven synthesis for automated feedback for introductory computer science assignments," Proceedings of Data Mining for Educational Assessment and Feedback (ASSESS 2014), 2014.

[34] S. Gross, B. Mokbel, B. Hammer and N. Pinkwart, "How to select an example? a comparison of selection strategies in example-based learning," International Conference on Intelligent Tutoring Systems, pp. 340-347, 2014.

[35] K. Zimmerman and C. R. Rupakheti, "An Automated Framework for Recommending Program Elements to Novices (N)," Automated Software Engineering (ASE), 30th IEEE/ACM International Conference on, pp. 283-288, 2015.

[36] P. Freeman, I. Watson and P. Denny, "Inferring Student Coding Goals Using Abstract Syntax Trees," International Conference on Case-Based Reasoning, pp. 139-153, 2016.

[37] T. Barnes, and J. Stamper, "Toward automatic hint generation for logic proof tutoring using historical student data," International Conference on Intelligent Tutoring Systems, pp. 373-382, 2008.

[38] D. Fossati, B. D. Eugenio, S. Ohlsson, C. W. Brown, L. Chen and D. G. Cosejo, "I learn from you, you learn from me: How to make iList learn from students," Journal of AIED, pp. 491-498, 2009.

[39] D. Fossati, B. D. Eugenio, S. Ohlsson, C. Brown and L. Chen, "Data driven automatic feedback generation in the iList intelligent tutoring system," Journal of Technology, Instruction, Cognition and Learning 10(1), pp. 5-26, 2015.

[40] H. Keuning, "Strategy-based feedback for imperative programming exercises," dspace.ou.nl/bitstream/1820/5388/1, 2014. [Online]: Available: dspace.ou.nl/bitstream/1820/5388/1/INF_20140 617_Keuning.pdf. [Accessed: July. 20, 2015].

[41] K. Rivers and K. R. Koedinger, "Automatic generation of programming feedback: A data-driven approach," The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013), vol. 50, 2013.

[42] K. Rivers and K. R. Koedinger, "Automating hint generation with solution space path construction," International Conference on Intelligent Tutoring Systems, pp. 329-339, 2014.

[43] A.Nguyen, C. Piech, J. Huang and L. Guibas, "Codewebs: scalable homework search for massive open online programming courses," Proceedings of the 23rd international conference on World Wide Web, pp. 491-502, 2014.

[44] T. Akutsu, "Tree edit distance problems: Algorithms and applications to bioinformatics," Journal of IEICE transactions on information and systems 93(2), pp. 208-218, 2010.

[45] L. Luo and Q. Zeng, "SolMiner: mining distinct solutions in programs," Proceedings of the 38th International Conference on Software Engineering Companion, pp. 481-490, 2016.

[46] R. Hosseini, A. Vihavainen and P. Brusilovsky, “Exploring Problem Solving Paths in a Java Programming Course,” Psychology of Programming Interest Group Conference, PPIG, pp. 65–76, 2014.

[47] P. Blikstein, M. Worsley, C. Piech, M. Sahami, S. Cooper and D. Koller, "Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming," Journal of the Learning Sciences 23(4), pp. 561-599, 2014.

[48] A.Vihavainen, M. Luukkainen and P. Ihantola, "Analysis of source code snapshot granularity levels," Proceedings of the 15th Annual

Conference on Information technology education, pp. 21-26, 2014.

[49] D. Hovemeyer, David, A. Hellas, A. Petersen and J. Spacco, "Control-Flow-Only Abstract Syntax Trees for Analyzing Students' Programming Progress," Proceedings of the 2016 ACM Conference on International Computing Education Research, pp. 63-72, 2016.

[50] A.Vihavainen, M. Luukkainen and J. Kurhila, "Multi-faceted support for MOOC in programming," Proceedings of the 13th annual conference on Information technology education, pp. 171-176, 2012.

[51] B. Hieu and N. Y, “Analysis Techniques for Data-Driven Hint Generation for Code-Writing Intelligent Tutoring Systems,” IPASJ International Journal of Information Technology (IIJIT), 5(3), 2017.

[52] Hieu Bui. A Classification of Data-Driven Hint Generation Techniques for Code-Writing Intelligent Tutoring Systems. American Journal of Computer Science and Information Engineering. Vol. 4, No. 2, 2017, pp. 16-23.