Automated Data-Driven Hint Generation in Intelligent Tutoring Systems for Code-Writing: On the Road of Future Research

https://doi.org/10.3991/ijet.v13i09.8023

Bui Trong Hieu(✉)
Ho Chi Minh City University of Transport, Ho Chi Minh City, Vietnam
Asia e University, Kuala Lumpur, Malaysia
hieu.bui@ut.edu.vn; hieubt@hcmutrans.edu.vn

S.M.F.D Syed Mustapha
Taif University, Taif, Saudi Arabia

Abstract—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 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. 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. Our goal in this paper is to review and classify analysis techniques that are requested to generate data-driven hints in ITSs for programming. This work also aims equally to identify the possible future directions in this research field.

Keywords—intelligent tutoring systems, data-driven hint generation, programming exercises

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]. As shown by VanLehn [3], an ITS 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, 15, 16, 17, 18, 19, 20, 21, 22]. According to [22], ITSs can provide personalized feedback to students automatically, but they can take large amounts of time and expert knowledge to build, especially when determining how to give students hints. Data-driven approaches can be used to provide personalized next-step hints automatically and at scale, by mining previous students' solutions. 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]. The primary contributions of this paper are 1) a classification of ITSs for programming, 2) a review of current data-driven hint generation approaches for ITSs for code-writing and 3) a discussion of the challenges that need to be addressed before we can expect to generate hints for data-driven ITSs for code-writing.

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 [23].

According to [24], an Intelligent Tutoring System (ITS) is a computer system that provides immediate and customized instruction or feedback to learners. The classical architecture of an ITS includes the following four components (Figure 2) [25, 26, 27, 28, 65].

- 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.

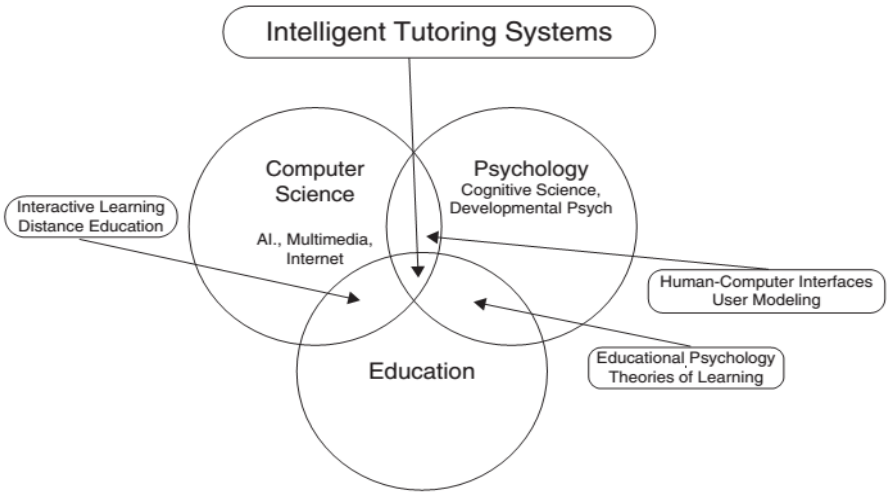

Fig. 1. The development of an ITS using methods and instruments from three different domains

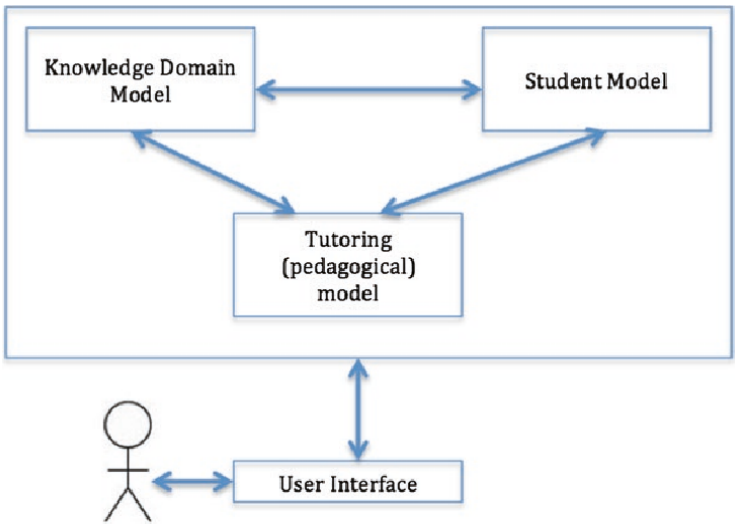

Fig. 2. The typical architecture of an ITS

This traditional view of ITSs is still very accepted by the ITS community. However, recent studies stress functionality over structure [29, 30, 25, 7, 31], describing ITSs as having two main loops [29]: 1) the inner loop and 2) the outer loop (Figure 3) [25]. 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.

```
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;
                }
        }
```

Fig. 3. ITS Loops

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. 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.

According to [32], 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, ITS is now a fast growing area in academia and industry. We now turn our attention to some cutting-edge research on ITS in a specific learning domain: programming [33].

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 exercises. 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 Automated hint generation in ITSs for code-writing

As demonstrated by [1], these non-data-driven techniques are including plan libraries, program transformation, constraint-based models, strategy- based models.
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 [34], ITSs for code-writing that focus on the process of solving an exercise are still rare or have limitations: some targeted for declarative programming [35, 6], which is less flexible because they do not support exercises that can be solved by multiple algorithms [36, 37], or only support a static, pre-defined process [38]. Furthermore, it often requires substantial work to add new exercises [39] and tutors can be difficult to adapt by a teacher.

ASK-ELLE [11] is an ITS 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 [33], 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 programming 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. It is worth noting that a range of non-data-driven techniques can be used to generate feedback and hints for programming exercises automatically [22].

As mentioned by [7], data-driven ITS 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. Successful solutions from the past can be used to provide feedback and hints for students in the present, which circumvents the need to create an expert model. A data-driven tutoring system can be bootstrapped by experts providing missing data. The data-driven approach has proven to work well in combination with artificial intelligence and machine-learning techniques for learning an expert model by demonstration [40].

3 Automated data-driven hint generation approaches in ITSs for code-writing

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. In recent years, 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 [41, 42]. In this research work, we focus on the hint generation for code correctness.

3.1 Program synthesis approach

In [39], the author 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 [43], this system requires experts/teachers to define an error model specific to each programming exercise, and only supports a subset of Python. In [6], the authors 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. [44] 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. As mentioned by [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.

Rolim et al. [20] take an example-based approach to learn code fixes as abstract syntax tree transformations from pairs of incorrect and correct student submissions. However, while this approach requires far less engineering effort, it may fail to generate hints, especially when a student's program is not close to a correct solution [17].

Head et al. [19] introduce a mixed-initiative approach which combines teacher expertise with data-driven program synthesis techniques. Their work has demonstrated

how program analysis and synthesis can be used as an aid for a teacher to scale feedback grounded in their deep domain knowledge. While scaling up teacher effort, these systems still require teachers to manually review and write hints for incorrect student work [45].

Suzuki et al. [45] explore a design space of hints that can be automatically generated from code transformations learned by program synthesis. Authors' ultimate goal is to adapt the strategies that a human teacher employs to automated hints driven by program synthesis. They identified five types of teacher hints that can also be generated by program synthesis. These hints describe transformations, locations, data, behavior, and examples. Their hints rely on the capabilities of program synthesis techniques to discover code transformations that fix incorrect code. As noted by the authors, while such techniques have been demonstrated on short assignments in introductory programming classrooms, in the future it may be possible to learn generalizable fixes for larger, more complex programs.

In [17], the authors present a robust hint generation system that extends the coverage of the program synthesis based approach using two complementary techniques. A syntax checker detects common syntax misconception errors in individual subexpressions to guide students to partial solutions that can be evaluated for the semantic correctness. A program synthesis based approach is then used to generate hints for almost-correct programs. If the program synthesis-based approach fails, a case analyzer detects missing program branches to guide students to partial solutions with reasonable structures. According to the authors, their experience suggests several ways that the system could be improved further.

3.2 Cluster based techniques

Gross et al. [46, 47] 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.

In [13], the authors 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 ITS 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).

Kaleeswaran et al. [48] propose a semi-supervised technique for feedback generation. This technique clusters the solutions based on the strategies to solve it. Then instructors manually label in each cluster one correct submission. They formally validate the incorrect solutions against the correct one. However, as noted by the authors,

there are many possible directions to improve clustering and verification by designing sophisticated algorithms.

Gulwani et al. [49] present a novel technique for clustering and repairing introductory programming assignments. They cluster correct submissions using variable traces computed for different inputs. Then, a representative submission from each cluster is selected as a reference solution. An incorrect submission is compared to each reference solution using variables traces, and some repairs are computed. The technique provides personalized feedback using the reference solution with the least number of repairs. However, a problem of this technique is that it requires inputs that are not easy to provide to trigger all possible errors. Variable traces are compared as a whole, so it needs a reference solution per any possible variation of a given assignment. Furthermore, the technique is not able to deal with infinite loops and submissions with multiple methods [50].

3.3 Recommendation approach

In [51], the authors 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 abstract syntax tree (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. With the example-based learning (EBL) strategy, Chaturvedi [52] presents a framework called Example Recommendation System (ERS) that is built upon EBL and that uses state-of-the-art mining algorithms in order to recommend a focused, organized and customized list of worked-out examples with the overall objective of increasing the likelihood of student success in the ITS's domain. However, as noted by the author, the limitation of algorithms used in this system is manual construction of regular expressions (RE) by experts.

3.4 Case-based reasoning approach

Freeman, et al. [53] 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 approaches

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 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 ITS that teaches programming [33].

In [54], the authors 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. [55, 56] implemented Hint Factory in the iList tutor that helps students learn linked list, a demanding topic in information technology and computer science education. In [56], the authors also concluded that their tutor produced equivalent learning gains to a human tutor.

Using the Hint Factory approach, Jin et al. [57] 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, in [58], the author 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.

In [7], the authors 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.

As demonstrated in [7], ITAP (Intelligent Teaching Assistant for Programming) 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, as noted by the authors, the path construction algorithm could be modified to further improve the performance. However, according to [51], one major pitfall of AST representations of source code is the loss of behavioral information.

Price et al. [12] present a new data-driven algorithm (CTD: Contextual Tree Decomposition), 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.

More recently, Price et al. [59] present iSnap, an extension to the Snap programming environment which adds some key features of ITSs, including detailed logging and automatically generated hints. They share results from a pilot study of iSnap, indicating that students are generally willing to use hints and that hints can create positive outcomes. Hints in iSnap are generated using the CTD algorithm. As noted by the authors, the study revealed several remaining challenges for the CTD algorithm and the presentation of iSnap hints.

3.6 Summary

In summary, there has been two board lines of research proposed for data-driven generating hints in ITSs for code-writing: program synthesis based and Hint Factory based. However, according to [60], there are two major drawbacks of program synthesis based approaches. First, an instructor must manually provide error models for each problem. Second, scalability is a big issue, especially with larger programs. 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.

4 Conclusion and future research

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 (snapshot of source code, a set of features, the actual code of program); 2) immediate representation of computer programs (AST, source code); 3) extracting distinct solutions of a programming exercise (preprocessing); 4) granularity of the code state used; 5) automatically modeling solution steps and 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. 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, according to [61], the snapshots are captured every time students compiled or saved their

code, but this is not an accurate representation of a unit of work (e.g., a line of code, a statement of source code)

2. **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.
3. **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. 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 [62].
4. **Programming exercises supported by data-driven ITSs code-writing.** It is important that a data-driven ITS for code-writing provides a collection of programming exercises covering an introductory programming course syllabus. Nevertheless, these programming exercises are generally stored in proprietary systems for their own use. According to [63], in general, two issues were detected that can hinder the proliferation of ITSs for code-writing: the lack of content standards for describing programming exercises and to communicate with other ITSs for code-writing.
5. **Programming language.** 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++ programming.
6. **Integrate data-driven ITSs for code-writing into curriculum.** As noted by Rivers [64], data-driven ITSs for programming has been expanding as a subfield of ITSs over the past few years, with many different researchers creating new techniques to automatically generate hints. However, most of the systems (including theirs) have only been evaluated on collected student problem-solving traces, and the ones that are being tested on real students are implemented in online learning environments such as MOOCs (massive open online courses), not in individual classrooms. In the context of curriculum and real classroom in an ITS, this indicates that there is significant room for improvement in the field of data-driven ITS for code-writing.

5 References

[1] Le.N.T. "Analysis Techniques for Feedback-Based Educational Systems for Programming". *4th International Conference on Computer Science, Applied Mathematics and Applications*, Vienna, Austria, 2016, pp. 141-152.

[2] R. Chughtai, S. Zhang and S. Craig. "*Usability evaluation of intelligent tutoring system*". Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 59, no. 1, 2015, pp. 367-371. https://doi.org/10.1177/1541931215591076

[3] K. VanLehn. "The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems". *Educational Psychologist*, vol. 46, no. 4, pp. 197-221, 2011. https://doi.org/10.1080/00461520.2011.611369

[4] M. Gomez-Albarran. "The Teaching and Learning of Programming: A Survey of Supporting Software Tools". *The Computer Journal*, vol. 48, no. 2, pp. 130-144, 2005. https://doi.org/10.1093/comjnl/bxh080

[5] Jin, Wei, T. Barnes, J. Stamper, M. J. Eagle, Matthew W. Johnson, and L. Lehmann. "Program representation for automatic hint generation for a data-driven novice programming tutor". *International Conference on Intelligent Tutoring Systems*, 2012, pp. 304-309. https://doi.org/10.1007/978-3-642-30950-2_40

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

[7] K. Rivers and K. Koedinger. "Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor". *International Journal of Artificial Intelligence in Education*, vol. 27, no. 1, pp. 37-64, 2015. https://doi.org/10.1007/s40593-015-0070-z

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

[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*, 2015, pp. 195-204.

[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*, 2016, pp. 191-198.

[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, pp. 183-190.

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

[15] S. Chow, K. Yace, I. Koprinska and J. Curran "Automated Data-Driven Hints for Computer Programming Students". *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, 2017, ACM, pp. 5-10.

[16] T. Lazar, A. Sadikov and I. Bratko "Rewrite Rules for Debugging Student Programs in Programming Tutors". *IEEE Transactions on Learning Technologies*, 2017. https://doi.org/10.1109/TLT.2017.2743701

[17] M. P. Phothilimthana and S. Sridhara "High-Coverage Hint Generation for Massive Courses: Do Automated Hints Help CS1 Students?". *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 2017, ACM, pp. 182-187.

[18] J. Yi, U. Z. Ahmed, A. Karkare, S. H. Tan and A. Roychoudhury "A feasibility study of using automated program repair for introductory programming assignments". *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn,*

Germany, September 4-8, (ESEC/FSE'17), 2017, 12 pages https://doi.org/10.1145/3106237.3106262

[19] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni and B. Hartmann "Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis". *Proceedings of the Fourth ACM Conference on Learning@ Scale*, 2017, pp. 89-98.

[20] R. Rolim, G. Soares, L. D'Antoni, O. Polozov, S. Gulwani, R. Gheyi and B. Hartmann "Learning syntactic program transformations from examples". *Proceedings of the 39th International Conference on Software Engineering*, 2017, IEEE Press, pp. 404-415.

[21] B. Paaßen, B. Hammer, T. W. Price, T. Barnes, S. Gross and N. Pinkwart. "The Continuous Hint Factory-Providing Hints in Vast and Sparsely Populated Edit Distance Spaces". *Journal of Educational Datamining,* 2017.

[22] K. Rivers "Automated Data-Driven Hint Generation for Learning Programming". PhD thesis, Human-Computer Interaction Institute, School of Computer Science, Carnegie Mellon University, USA, 2017.

[23] W. B. Park. (2008). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. [On-line]. Available: https://libgen.pw/download.php?id=313548 [May 20, 2016].

[24] L. R. Maria and T. T. Chen. "Digital creativity: Research themes and framework". *Journal of Computers in Human Behavior*, 42, pp. 12-19, 2015. https://doi.org/10.1016/j.chb.2014.04.001

[25] 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. https://doi.org/10.1109/TLT.2013.17

[26] W. D. Samanthi. "Intelligent tutoring system for learning PHP". PhD thesis, Queensland University of Technology, Australia, 2013. [On-line]: Available: http://eprints.qut.edu.au/63202/1/Dinesha%20Samanthi_Weragama_Thesis.pdf. [Feb. 12, 2014].

[27] H. Budi. "Incorporating anchored learning in a C# intelligent tutoring system". PhD thesis, Queensland University of Technology, Australia, 2014. [On-line]: Available: http://eprints.qut.edu.au/78834/1/Budi_Hartanto_Thesis.pdf. [Dec. 12, 2014].

[28] C. Konstantina and M. Virvou. (2015). *Advances in Personalized Web-Based Education*. pp. 1-24.

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

[30] G. Alex. "Ask-Elle: a Haskell Tutor". PhD thesis, Open Univeristy, Utrecht University, The Netherlands, 2012. [On-line]: Available: http://www.botkes.nl/wp-content/uploads/HaskellTutor.pdf. [Dec. 20, 2013].

[31] 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.

[32] 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.

[33] M. T. Irfan and V. N. Gudivada. (2016). *Handbook of Statistics*. [On-line]. Vol. 35. Available: [May 25, 2016]. https://doi.org/10.1016/bs.host.2016.07.008

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

[35] 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. https://doi.org/10.1016/j.ijhcs.2004.02.001

[36] J. R. Anderson and E. Skwarecki. "The automated tutoring of introductory computer programming". *Communications of the ACM*, 29(9), pp. 842-849, 1986. https://doi.org/10.1145/6592.6593

[37] 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. https://doi.org/10.1080/1049482940040202

[38] 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*, 2014, pp. 318-328. https://doi.org/10.1007/978-3-319-07221-0_40

[39] R. Singh. "Accessible Programming using Program Synthesis". PhD thesis. Massachusetts Institute of Technology, USA, 2014. [On-line]: Available: http://people.csail.mit.edu/rishabh/papers/rishabh_thesis.pdf. [Dec. 30, 2015].

[40] Heeren, B., & Jeuring. "An extensible domain-specific language for describing problem-solving procedures". Internet: http://www.cs.uu.nl/research/techreps/repo/CS-2017/2017-007.pdf, Jul., 2017 [Aug. 23, 2017]

[41] Choudhury, R. R., Yin, H., & Fox, A. "Scale-Driven Automatic Hint Generation for Coding Style". *International Conference on Intelligent Tutoring Systems, 2016,* pp. 122-132.

[42] Wiese, E. S., Yen, M., Chen, A., Santos, L. A., & Fox, A. "Teaching Students to Recognize and Implement Good Coding Style". *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, 2017, pp. 41-50.

[43] Terman, S. " GroverCode: Code Canonicalization and Clustering Applied to Grading", Phd thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, US, 2016. [On-line]. Available: http://up.csail.mit.edu/other-pubs/seterman-thesis.pdf. [Oct. 10, 2016].

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

[45] Suzuki, R., Soares, G., Glassman, E., Head, A., D'Antoni, L., & Hartmann, B. "Exploring the Design Space of Automatically Synthesized Hints for Introductory Programming Assignments". *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2017, pp. 2951-2958. https://doi.org/10.1145/3027063.3053187

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

[47] Gross, S., Mokbel, B., Hammer, B., & Pinkwart, N. "How to select an example? a comparison of selection strategies in example-based learning". *International Conference on Intelligent Tutoring Systems*, 2014, pp. 340-347. https://doi.org/10.1007/978-3-319-07221-0_42

[48] Kaleeswaran, S., Santhiar, A., Kanade, A., & Gulwani, S. "Semi-supervised verified feedback generation". *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 739-750. https://doi.org/10.1145/2950290.2950363

[49] Gulwani, S., Radiček, I., & Zuleger, F. "Automated Clustering and Program Repair for Introductory Programming Assignments". *arXiv preprint arXiv:1603.03165,* 2016.

[50] Marin, V. J., Pereira, T., Sridharan, S., & Rivero, C. R. "Automated Personalized Feedback in Introductory Java Programming MOOCs". *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, 2017, pp. 1259-1270.

[51] Zimmerman, K., & Rupakheti, C. R. "An Automated Framework for Recommending Program Elements to Novices (N)". *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, 2015, pp. 283-288.
[52] Chaturvedi, R. "Task-based Example Miner for Intelligent Tutoring Systems". Doctoral dissertation, *University of Windsor,* Windsor, Ontario, Canada, 2016. [On-line]. Available: http://scholar.uwindsor.ca/etd/5790/. [Nov. 20, 2016].
[53] Freeman, P., Watson, I., & Denny, P. "Inferring Student Coding Goals Using Abstract Syntax Trees". *International Conference on Case-Based Reasoning*, 2016, pp. 139-153.
[54] Barnes, T., & Stamper, J. "Toward automatic hint generation for logic proof tutoring using historical student data". *International Conference on Intelligent Tutoring Systems* , 2008, pp. 373-382. https://doi.org/10.1007/978-3-540-69132-7_41
[55] Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., Chen, L., & Cosejo, D. "I learn from you, you learn from me: How to make iList learn from students". *Proceedings of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, 2009, pp. 491-498.
[56] Fossati, D., Di Eugenio, B., Ohlsson, S. T. E. L. L. A. N., Brown, C., & Chen, L. "Data driven automatic feedback generation in the iList intelligent tutoring system". *Technology, Instruction, Cognition and Learning*, *10*(1), 5-26, 2015.
[57] Jin, W., Barnes, T., Stamper, J., Eagle, M. J., Johnson, M. W., & Lehmann, L. "Program representation for automatic hint generation for a data-driven novice programming tutor". *International Conference on Intelligent Tutoring Systems*, 2012, pp. 304-309. https://doi.org/10.1007/978-3-642-30950-2_40
[58] Keuning, H. "Strategy-based feedback for imperative programming exercises". PhD thesis, Utrecht University, Utrecht, The Netherlands, 2014. [On-line]. Available: http://dspace.learningnetworks.org/handle/1820/5388. [Nov 11, 2015].
[59] Price, T. W., Dong, Y., & Lipovac, D. "iSnap: Towards Intelligent Tutoring in Novice Programming Environments". *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education,* 2017*,* pp. 483-488. https://doi.org/10.1145/3017680.3017762
[60] Wang, K., Lin, B., Rettig, B., Pardi, P., & Singh, R. "Data-Driven Feedback Generator for Online Programing Courses". *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, 2017, pp. 257-260.
[61] Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. "Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming". *Journal of the Learning Sciences*, 23(4), 561-599, 2014. https://doi.org/10.1080/10508406.2014.954750
[62] Luo, L., & Zeng, Q. "SolMiner: mining distinct solutions in programs". *Proceedings of the 38th International Conference on Software Engineering Companion*, 2016, pp. 481-490.
[63] Queirós, R., & Leal, J. P. "A survey of e-learning content aggregation standards". *International Conference on Web-Based Learning*, 2014, pp. 204-214.
[64] Rivers, K. "Designing a Data-Driven Tutor Authoring Tool for CS Educators". *Proceedings of the eleventh annual International Conference on International Computing Education Research*, 2015, pp. 277-278.
[65] Syed Mustapha, S.M.F.D. "Building Learning System for Content Knowledge and Social Knowledge". *International Journal of Emerging Technologies in Learning (iJET)*, *13*(01), pp. 4-22, 2018. https://doi.org/10.3991/ijet.v13i01.6912

6 Authors

Bui Trong Hieu is a PhD student in School of Science and Technology, Asia e Univeristy, Kuala Lumpur, Malaysia. He works at the Information Technology Faculty of the Ho Chi Minh City University of Transport, Vietnam.

S.M.F.D Syed Mustapha is a professor in Computer Science Department, College of Computers and Information Technology, Taif University, Saudi Arabia. His main research interest is on building intelligent techniques through knowledge modelling for learning in which he had applied in various domains such as rheology, inorganic chemistry, social communication and community of practice. He received his PhD and MPhil from University of Wales, UK and Bachelor of Science (Computer Science) from University of Texas, USA.

Article submitted 26 November 2017. Resubmitted 09 May 2018. Final acceptance 11 May 2018. Final version published as submitted by the authors.