Providing attempt-based help and scaffolding with an automatically generated trace table (Independent Study Summer '20)

Zak Risha
University of Pittsburgh
Pittsburgh, Pennsylvania
zak.risha@pitt.edu

ABSTRACT

In this study we introduce a new tool, an automatically generated trace table for Java, and investigate its use as a form of help in an existing tutoring system. Findings from a classroom study suggest the scaffolding provided by the trace table is a plausible alternative to other forms of help such as hints, most notably, increases in performance and persistence and lower task difficulty. Adoption of the on-demand tool proved to be low, however recommendation utilizing a student model proved to be effective in increasing engagement. Based on usage data we propose future implications for adaptive design and student modeling.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

ITS, Problem generation, scaffolding

ACM Reference Format:
Zak Risha. 2018. Providing attempt-based help and scaffolding with an automatically generated trace table (Independent Study Summer '20). In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

A common, valued attribute of intelligent tutoring systems (ITS) is their ability to break problems into steps, providing feedback and hints throughout the entire process. Such a feature is often referred to as an inner loop and is one of the defining characteristics of intelligent tutor behavior [27]. These step-by-step tasks not only result in a better learning experience for students, allowing them to catch errors and provide scaffolding throughout the process, but also provide a better model of the student's knowledge. This approach does have a few shortcomings, as having students complete each step can slow down their productivity. Additionally, breaking problems into steps can be a time consuming endeavor, often requiring expertise to decipher how to break down a task and modify the tutor. Systems like CTAT [1] have tried to alleviate the need for programming expertise by providing an interface for content experts to create their own step-by-step problems. While this is useful, it still requires human expertise and takes time to craft the problem. Some domains, such a programming, offer affordances that can automate this process entirely, allowing for easy integration of step-by-step tasks.

1.1 Automation in ITS

While ITS has often been noted for strong learning gains, a common stumbling block to deployment is the amount of time needed to construct these systems [7] [8]. Therefore, it is not much surprise that researchers have sought to automate aspects of the creation process. One of the notable areas has been the automation of hints for each sequential step of a problem. Efforts like Bootstrapping Novice Data [14] and Hint Factory [4] relied on prior student log data to generate hints based on student behaviors. Rivers and Koedinger attempted to extend such an approach to tackle the open-ended, problem-solving domain of program construction where there are infinite possible states in a problem [21]. Recent research has gone a step further and studied the impact of such automated hints with mixed results, either noting performance gains but requiring additional self-explanations for post-test improvements [13] or having no impact on performance or learning [19]. While these data-driven approaches have had some success, they come with the limitation of requiring sufficient data and a robust enough model to guarantee coverage.

Alternatively, researchers have tried to rely on particular affordances of a domain to automatically generate steps, hints, and visualizations. O'Rourke [17] [18] mapped algorithms to problem solving processes in the domain of fractions and algebra, allowing for the automatic generation of not only problems with steps, but also hints and feedback. In O'Rouke's case, an expert was needed to map a programming algorithm to a problem space, but after was able to consistently generate steps and hints for similar problems within the selected domain. Such approaches helps reduce the burden of constructing these systems and providing more varied content.

1.2 Automation in Programming Problems

Code offers many affordances that can makes expert mapping unnecessary, covering a broader range of problems without additional

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/10.1145/1122445.1122456

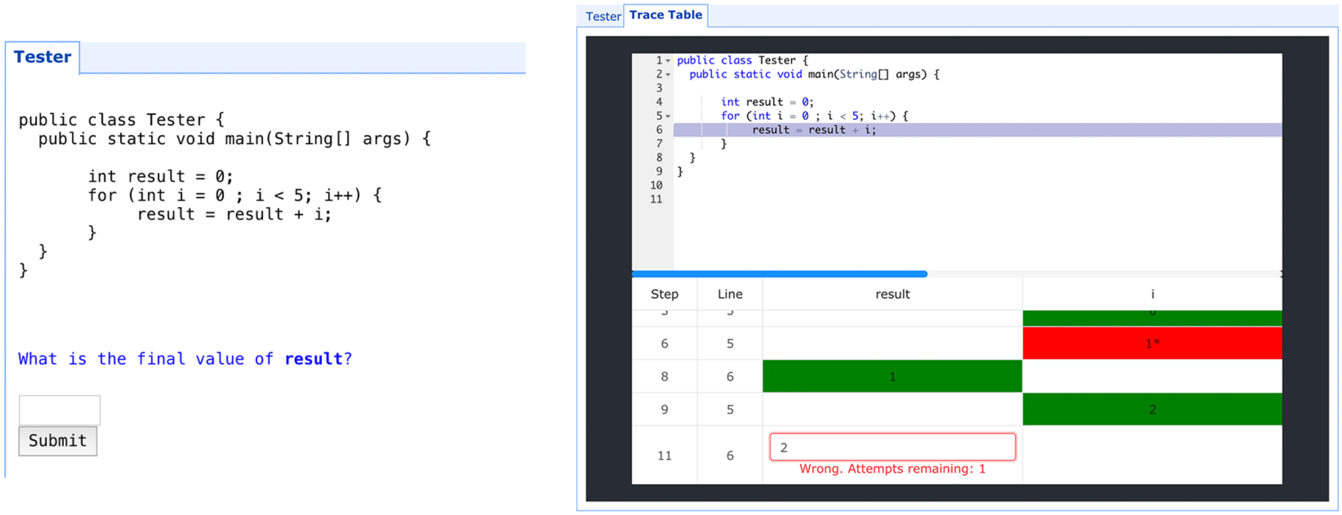

Figure 1: A Quizjet Problem (left) and a Trace Table (right) generated from the same program

modifications to the system. Two notable approaches have been applied to create program visualization tools. JSVEE [26] developed a transpiler based on the static analysis of code, producing an output that could be utilized by a visualization tool. This method allows for more detailed evaluation at the expression level and possessed the ability to handle a wide array of problems, but similar to O'Rourke's approach, required the manual development of a transpiler to interpret the code. Guo also built a visualization tool, the Python Tutor [9], but relied on *execution traces* produced by debugging instruments often found in programming languages, creating line based visualizations. While not as detailed an evaluation, this approach scales more easily to multiple programming languages, as Python Tutor [1] now supports five languages.

While these affordances of code have been used to create visualizations, they can also be used to generated problems for a more active experience for users. These *execution traces* produced by debuggers contain sufficient information to break a problem into steps. Our approach is to map these resulting traces to an existing pedagogical strategy used when teaching programming—the trace table. A trace table is a technique which simulates the execution of a program by tracking the changes of variables through each step of the code. Xie et al. [30] acknowledged the existence of the aforementioned tools [9, 26] for visualization of the program state, but described them as passive, lacking active engagement. Due to the absence of any trace table software, they investigated the value of teaching a paper based trace table strategy to novel students of programming, finding that an experiment group taught the strategy scored 15% higher on an assessment. We investigate automatically providing such a strategy via software to alleviate the burden on students of learning and applying the strategy with paper. In addition, this turns the trace table into an activity easily integrated into learning systems like intelligent tutors.

[1] http://pythontutor.com/

2 METHOD

Our approach automatically generates an interactive table from a given piece of source code. The resulting trace table software can help students navigate a program's execution and provide immediate feedback of the correct change in a given step. In this manner, students are actively engaged as they track and predict the execution of a program step-by-step, providing an automatically constructed inner-loop for students that helps them diagnose errors. We built the trace table using web technologies, more specifically, a Javascript library for single page web applications and a server that utilizes execution traces generated from the Python Tutor backend [9]. The interface shows the code being traced, highlighting the current line being executed. Variables and output are columns in the table with the Step and Line acting as row indexes. The software tracks each change of a variable and adds new columns and rows dynamically as variables are introduced or values changed. Students are prompted for a variable value at a given step of execution and asked to enter it into the table. Each task is provided with up to three attempts, giving the correct answer only after exhausting all attempts. To address domain modeling, knowledge components are mapped to each line automatically using the JavaParser which utilizes the abstract syntax tree to parse the code [10].

While an auto-generated trace table could have many pedagogical applications, we investigate its use as a form of help by integrating it into an existing system, Quizjet [11], as a supplementary activity to provide learner support. Quizjet consists of prediction tasks where users must respond with the final value or output of a parameterized program. Upon failure, students are provided the correct answer but the problem parameters will be altered in future attempts. These problems do not utilize an inner loop and consist of a single observable step. Similar to JSVEE and Python tutor, the trace table is designed to be incorporated and embedded into various existing systems. As the trace table is built with web technologies and requires only source code as an input, it was easily incorporated into the existing QuizJet system. The current library only requires inclusion of a JavaScript and CSS file and provides an API to render and configure the trace table on a web page. Students could access the trace table on-demand after attempting a problem in Quizjet via a button labeled "Trace this code" which would trace the corresponding QuizJet problem.

2.1 Help in Tutoring Systems

For our study, we situate the trace table as a method of help for struggling or curious students. Research has investigated the effectiveness of on-demand help, including, but not limited to, how the help is provided, when it is accessed, and how students utilize the help [2]. After noting undesirable behavior, such as help avoidance and abuse, Roll et al. created the Help Tutor [22] which attempted to address how students engage with help resources by creating a help-seeking model that provides metacognitive feedback. While the approach did not result in notable learning outcomes, it did highlight ways to aid students in using help more deliberately. The association with help and learning gains has not always been significant, leading some to conclude that *when* and *how* help occurs plays a significant role in effectiveness [2]. Future analysis of log data by Roll et al. suggested that principle-based hints during medium level of mastery were most effective [23]. Additionally, studies have questioned the degree to which hints are useful, noting that reattempting a problem, rather than obtaining a hint, may be better for learning [24]. Such findings bring forefront the question of *how* a system should offer help, suggesting more approaches should be explored. Studies have noted the potential of replacing or augmenting hints with more interactive activities such as self explanations or scaffolding [20] [6] [2] [13]. As these alternative approaches could offer less support, it is also important to consider persistence, a valued trait for learners in a system. As research works to detect non-persistence in systems [5, 29], help should be able to serve as a sufficient intervention to prevent students from abandoning problems when encountering difficulty. Overall, more research needs to be performed to conclude how *help* can help most effectively in intelligent tutoring systems.

Our attempt-based approach focuses on actively engaging students through the help process. In lieu of giving a hint, the system provides an alternative activity that breaks down the given task. As Quizjet's problems are single step and not already broken down like some tutors, we can easily integrate this method into this system. Quizjet's single step can offer a way for students to progress through the system faster and solve more problems, but, when struggling, students likely need more support and thus can use the trace table for scaffolding. Therefore, the trace table acts as an alternative, interactive method of help which actively engages the student in a task as opposed to providing a principle-based hint. By using the trace table, students can identify the specific steps where they have made a misassumption and also better track the execution of the program. Likewise, the broken down tasks would also help the system diagnose areas where a student may be deficient. This also is an attempt-based approach that tries to capitalize on the correlation with learning gains seen when students attempt more problems, rather than given hints [24]. Feedback for attempts is limited to whether a solution is correct or incorrect and, after exhausting all allotted attempts, the correct answer. Since this form help is provided on-demand and to address the issue of when students should engage with the tool, we implement a popup recommendation in our experiment group based on the student model, recommending it after students fail on newer problems they have less experience with. This is not only an effort to combat help-avoidance but also ensure students are familiar with the help at their disposal.

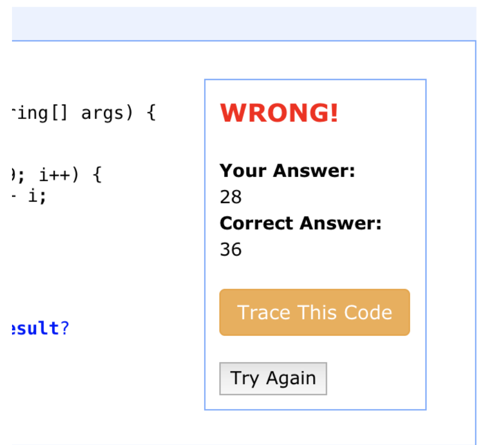

Figure 2: Feedback from a QuizJet attempt along with the button for on-demand trace table of same problem

3 STUDY AND RESULTS

A classroom study was conducted to pilot the trace table and investigate how it would be utilized by students for learning support in the existing QuizJet system. Subjects were enrolled in one of two sections in Fall 2019 or one section in Spring 2020 of an undergraduate Introductory Java course for information science students and non-majors. Students were not required to have prior programming experience. All students were provided an account to access the practice system at their own discretion. Students were randomly assigned to an experiment or control group that determined whether the system would recommend the trace table.

Over 101 unique Quizjet problems were attempted in the system, 66 of which included a trace table option. All problems were accessed through the Mastery Grids system [12] which provides an interface to navigate through all the available learning activities and includes an Open Learner Model (OLM) and social comparison elements. The OLM shows an estimation of a student's conceptual knowledge based on their performance, while the social comparison visualizes how the rest of the class is progressing on each topic. To incentive students to use the system, up to a 2% increase of their final grade was offered as extra credit for solving at least one problem per topic.

Of the 77 students enrolled in the courses, 50 attempted QuizJet problems and 35 made use of the trace table software. All enrolled users were randomized into two conditions and, of the 50 active users, 24 were from the control group and 26 were from the experiment group. A total of 297 trace tables were attempted, 256 of which completed, for 61 unique problems, generating a total of 2440 line attempts. There were a total of 4980 QuizJet submissions to 66 unique problems that supported trace tables, and, as questions were parameterized, the same problem could be reattempted multiple times by a user with differing answers.

Figure 3: Comparing QuizJet and Trace Table Accuracy in Classroom and Recitation studies

While the classroom study investigated long term use, a second study was conducted during Fall 2020 in a course recitation for an introductory computer science course to look at short-term effects and also impact on more challenging problems. This short study took place during the last two weeks of the semester where students solved 8 QuizJet problems and were able to use the trace table before or after each attempt. Completion of problems occured either in a lab or at a student's leisure at home. A total of 90 users participated in solving QuizJet problems with 80 making use of the trace table. A total of 1682 attempts were made of 8 unique Quizjet problems and 447 trace tables were accessed, 301 of which completed, to create a total of 5093 line attempts. Participation was entirely optional and was presented as a method to prepare for the upcoming course final.

3.1 Trace Table Difficulty

We first wanted to investigate the impact of breaking QuizJet problems into multiple steps had on learners, more specifically, the effect on the difficulty of the task inferred from student error rates. To eliminate some of the noise in the data from easier problems, we looked at the tougher half of problems based on difficulty, i.e., the mean error rate. Mean error rate of trace table problems was low at 18.1% with over 47.2% of the tables reporting 100% accuracy as opposed to Quizjet with 35.9% mean error rate and 39.7% with no errors. We are reluctant to report statistical tests directly comparing accuracy of Quizjet problems and Trace table problems due to their differing structure. While difficulty was challenging to compare between both sets of problems, these findings suggest that merely breaking a problem down reduced the error rate in a classroom setting. However, results differed when looking at recitation data, as shown in Figure 3. As opposed to the classroom setting, the recitation took place in a more condensed time frame with more challenging problems to prepare for a final examination. Evidence suggesting the trace table was less difficult was not as apparent, and instances of no errors were much more rare.

3.2 Accuracy Over Time

Similarly, we analyzed how students performed with each respective activity over time as the total number of opportunities increased. To accomplish this, we looked at users who had completed a minimum of four trace tables, resulting in 15 users, and calculated their accuracy per opportunities in both Quizjet and trace tables. Simple linear regression was performed to the examine the relationship between number of opportunities and overall error rate in both Quizjet and trace table problems. As shown in 4, for Quizjet there was a significant effect between opportunity number and error rate, the latter increased approximately 1 percent for every 5 opportunities. Conversely, the regression analysis found a significant, negative effect between opportunity number and error rate further described table 1. This result revealed that while error rate increased over time in Quizjet, the opposite occured in trace table problems, however this effect was less pronounced with a decrease of .35 percent for every 5 opportunities.

Table 1: Simple Regression by Number of Opportunities

Source	F	df	*B*	t	*p*	r^2
Quizjet	1410	1817	.0021	37.6	.001	.437
Trace Table	176.8	2104	-.0007	-13.3	.001	.078

Figure 4: QuizJet and Trace Table Error Rate over time

3.3 Remediation After Failing

We also wanted to investigate whether the trace table aided students in solving QuizJet problems they had previous failed on. Problems were parameterized so students would be given a different problem of similar structure following a wrong answer. We looked at instances where users fail and reattempt the problem without tracing compared to instances when user's fail, complete a trace table, then reattempted the problem. A chi square test was performed to examine the relationship between a user completing a trace table and answering correctly. Results indicated a significant relationship between completing a trace table and answering correctly $\chi^2(1, N = 671) = 5.8,\ p < .025\ \phi = 0.05\ OR = 1.85$. Likewise, we conducted the same analysis for the recitation with a significant, and stronger, relationship $\chi^2(1, N = 1062) = 28.5,\ p < .001,\ \phi = 0.77\ OR = 2.69$. This association gives evidence that the trace table was indeed helpful for students in solving problems they had failed on and this finding was strengthen in the recitation where students were 2.69 times more likely to answer correctly after tracing.

3.4 Persistence

Since students now had a helpful resource to use when failing a QuizJet problem, we wanted to examine persistence, conceptualized as instances where a problem was first attempted incorrectly but later solved. Abandonment, failure to solve an attempted problem, was infrequent, occurring only 30 times total in the classroom study. Due to the rarity of the event, we looked at just the QuizJet problems abandoned by users and calculated abandonment rates for those that were traced and those not traced. Users had significantly lower abandonment rates (M=0.02, SD=0.1) for problems where they traced the code than those in which they didn't (M=0.12, SD = 0.3), t(46) = 2.18, p < .05. While abandonment was rare, this finding suggests the trace table, when used, contributed to students persisting and eventually solving the problem.

3.5 Usage and Recommendations

Not all users made use of the trace table with 35 of 50 completing at least one problem, and 23 of those 35 completing 3 or more tables. We anticipated adoption to be an issue as prior research has noted help avoidance and worked to remedy it [3] [22]. To account for this, we attempted to promote use of trace table for struggling students in system by implementing a popup recommendation that triggered after failing a problem. The popup was dependent on the student model, and appeared only if a student lacked a mean 50% probability of knowing concepts associated with the problem. In the classroom study, the mean conversion rate of users was 65% indicating that the recommendation to table trace was fairly successful based on the model. A Welch's t-test showed that users receiving the popup recommendation (M=7.6, SD = 9.9) completed significantly more trace tables than users not shown a popup (M=2.7, SD = 5.1) t(50) = 2.27, p < .05. We desired to implement a third condition for randomized popups to evaluate the efficiency of adaptive recommendation but were unable to obtain sufficient subjects. Still, there was evidence the recommendation promoted the use of help in the system to warrant further investigation.

3.6 Other Observed Usage Behavior

There was little evidence of gaming the trace table, like the bottoming out of hints, which might be attributed to Quizjet's feedback alone often being sufficient to obtain a correct answer or the parameterized nature of the problems making gaming less effective. The few instances of gaming observed were actually due to problems with excessive number of steps. One such case involving an inner loop generated over 150 steps for the student to complete. Other similar idiosyncrasies were observed, such as input errors occurring over the specificity of data types (e.g., including a decimal and trailing zero in the answer). In one such instance a student expressed anger at the trace table in their entry for not accepting their answer as correct. Overall, such frustrations seemed more rare in the classroom study where usage occurred over a period of time and error rates were lower. These observations do highlight areas of potential improvement for the system to better adapt to both learners and content.

4 DISCUSSION

Our single session remediation study revealed a much lower degree of accuracy on the trace table. In this case, students had less time to familiarize themselves with the trace table strategy, suggesting the exercise might not be intuitive. Despite the low accuracy, students' performance seemed to benefit from the using the trace table. This could be due to the fact that an error ridden trace table still serves as a worked example, a finding in line with Shih et al.'s analysis of bottom out hints [25]. While still beneficial, this low accuracy presents an opportunity to adapt to new users by first modeling the trace table in the form of a worked example. By relying on a student model, the trace table could first visualize the execution before asking students to trace it themselves. This would create interleaved worked examples that might make students more efficient as observed in prior research [15, 16] and also could decrease student frustration. Such an adaptive design would be similar to step-based helped recently proposed in program construction tasks [28] and capitalize on higher levels of scaffolding while remaining more active than techniques like hints.

An additional problem for adaptation, the trace table was observed generating an excessive amount of steps for some types of problems resulting in a time consuming intervention that could impede learning. Loops in general produce repetitive tasks in tracing which is sometimes useful but often unnecessary. This calls for future adaptations that shorten the task by eliminating easier or repetitive steps based on the model on the student's knowledge. Simple steps like variable initialization are likely not pertinent in more advanced problems and could be skipped to reduce time on task. Similarly, loops could be traced only at crucial moments such as the start and end. However skipping too many steps could confuse learners in their efforts to trace the execution of the program, thus this adaptation must be implemented in an intuitive manner to alleviate frustrations. Additional feedback and instruction may be necessary to help address some of these issues of confusion suggested by the low accuracy in the recitation.

Additionally, when to promote this form of help was also investigated. Conversion rates were high for recommending the trace table after failing on problems with newer concepts and these prompts had a significant effect on how often the trace table was utilized. Still, use of trace table was low and one could reason students could learn from Quizjet's feedback of a correct answer. However, this could potentially lead to guessing or shallow learning as the parameterized problems are similar in structure. This feedback might also explain the low occurrence of abandonment, giving further motive to disable this type of feedback in future studies. While prompts for using the trace table lead to more engagement with this form of help, more research needs to be conducted to find when are the optimal moments for recommendation based on the student's knowledge and prior behavior. For example, prior research has indicated that the level of mastery or knowledge can impact the effectiveness of help. Such prompts could also utilize detection of wheel-spinning or non-persistence to redirect students efforts to a more scaffolded task.

5 LIMITATIONS

While this study noted improvements in student performance from using the trace table, further analysis needs to be conducted to investigate the trace table's relationship with learning gains of students. Unfortunately due to COVID-19, we were unable to access the pre and post test in time to conduct a thorough analysis of the effects. Additionally there were insufficient students for a third condition to evaluate the adaptive recommendation based on student knowledge and determine whether it would outperform a randomized prompt.

6 CONCLUSION

This study investigated the use of an automatically generated trace table as a help mechanism for students in a classroom and single session setting. Results suggest that automatically breaking down the task into steps decreased the difficulty, producing lower error rates. Error rates in Quizjet increased over time with more opportunities which we associate with increasing difficulty of problems, however the trace table's error rates decreased despite problems growing more difficult. Additionally, there was strong evidence the trace table aided in remediation by improving accuracy of students on failed problems in both classroom and recitation settings. Both these findings further amplify prior recommendations for researching other methods of help beyond the more traditional hints which have not always yielded significant results. Similarly, our analysis found that the trace table provided enough support to help students persist through difficult problems despite the active, attempt-based nature of the exercise.

REFERENCES

[1] Vincent Aleven, Bruce Mclaren, Jonathan Sewall, and Kenneth Koedinger. 2006. The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. 61–70. https://doi.org/10.1007/11774303_7

[2] Vincent Aleven, Ido Roll, Bruce M. McLaren, and Kenneth R. Koedinger. 2016. Help Helps, But Only So Much: Research on Help Seeking with Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education* 26, 1 (March 2016), 205–223. https://doi.org/10.1007/s40593-015-0089-1

[3] Vincent Aleven, Elmar Stahl, Silke Schworm, Frank Fischer, and Raven Wallace. 2003. Help seeking and help design in interactive learning environments. *Review of educational research* 73, 3 (2003), 277–320. Publisher: Sage Publications Sage CA: Thousand Oaks, CA.

[4] Tiffany Barnes and John Stamper. 2008. Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. In *Intelligent Tutoring Systems (Lecture Notes in Computer Science)*, Beverley P. Woolf, Esma Aïmeur, Roger Nkambou, and Susanne Lajoie (Eds.). Springer, Berlin, Heidelberg, 373–382. https://doi.org/10.1007/978-3-540-69132-7_41

[5] Anthony F. Botelho, Ashvini Varatharaj, Thanaporn Patikorn, Diana Doherty, Seth A. Adjei, and Joseph E. Beck. 2019. Developing Early Detectors of Student Attrition and Wheel Spinning Using Deep Learning. *IEEE Transactions on Learning Technologies* 12, 2 (April 2019), 158–170. https://doi.org/10.1109/TLT.2019.2912162 Conference Name: IEEE Transactions on Learning Technologies.

[6] Kirsten R. Butcher and Vincent Aleven. 2013. Using student interactions to foster rule–diagram mapping during problem solving in an intelligent tutoring system. *Journal of Educational Psychology* 105, 4 (2013), 988–1009. https://doi.org/10.1037/a0031756 Place: US Publisher: American Psychological Association.

[7] Albert T. Corbett, Kenneth R. Koedinger, and John R. Anderson. 1997. Intelligent Tutoring Systems. In *Handbook of Human-Computer Interaction*, Marting G. Helander, Thomas K. Landauer, and Prasad V. Prabhu (Eds.). North-Holland, Amsterdam, 849–874. https://doi.org/10.1016/B978-044481862-1.50103-5

[8] Jeremiah T Folsom-Kovarik, Sae Schatz, and Denise Nicholson. 2010. Plan ahead: Pricing ITS learner models. 47–54.

[9] Philip J. Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13)*. Association for Computing Machinery, Denver, Colorado, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[10] R. Hosseini and P. Brusilovsky. 2013. JavaParser: A fine-grain concept indexing tool for java problems. In *CEUR Workshop Proceedings*, Vol. 1009. University of Pittsburgh, 60–63. http://d.scholarship.pitt.edu/26270/

[11] I.-Han Hsiao, Peter Brusilovsky, and Sergey Sosnovsky. 2008. Web-based Parameterized Questions for Object-Oriented Programming. Association for the Advancement of Computing in Education (AACE), 3728–3735. https://www.learntechlib.org/primary/p/30206/

[12] Tomasz D. Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. 2014. Mastery Grids: An Open Source Social Educational Progress Visualization. In *Open Learning and Teaching in Educational Communities (Lecture Notes in Computer Science)*, Christoph Rensing, Sara de Freitas, Tobias Ley, and Pedro J. Muñoz-Merino (Eds.). Springer International Publishing, Cham, 235–248. https://doi.org/10.1007/978-3-319-11200-8_18

[13] Samiha Marwan, Joseph Jay Williams, and Thomas Price. 2019. An Evaluation of the Impact of Automated Programming Hints on Performance and Learning. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. Association for Computing Machinery, Toronto ON, Canada, 61–70. https://doi.org/10.1145/3291279.3339420

[14] Bruce M. McLaren, Kenneth R. Koedinger, Mike Schneider, Andreas Harrer, and Lars Bollen. 2004. Bootstrapping Novice Data: Semi-Automated Tutor Authoring Using Student Log Files. (Jan. 2004). https://doi.org/10.1184/R1/6469859.v1 Publisher: Carnegie Mellon University.

[15] Bruce M. McLaren, Sung-Joo Lim, and Kenneth R. Koedinger. 2008. When and how often should worked examples be given to students? New results and a summary of the current state of research. In *Proceedings of the 30th annual conference of the cognitive science society*. 2176–2181.

[16] Bruce M. McLaren, Tamara van Gog, Craig Ganoe, Michael Karabinos, and David Yaron. 2016. The efficiency of worked examples compared to erroneous examples, tutored problem solving, and problem solving in computer-based learning environments. *Computers in Human Behavior* 55 (Feb. 2016), 87–99. https://doi.org/10.1016/j.chb.2015.08.038

[17] Eleanor O'Rourke, Erik Andersen, Sumit Gulwani, and Zoran Popović. 2015. A Framework for Automatically Generating Interactive Instructional Scaffolding. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, Seoul, Republic of Korea, 1545–1554. https://doi.org/10.1145/2702123.2702580

[18] Eleanor O'Rourke, Eric Butler, Armando Díaz Tolentino, and Zoran Popović. 2019. Automatic Generation of Problems and Explanations for an Intelligent Algebra Tutor. In *Artificial Intelligence in Education (Lecture Notes in Computer Science)*, Seiji Isotani, Eva Millán, Amy Ogan, Peter Hastings, Bruce McLaren, and Rose Luckin (Eds.). Springer International Publishing, Cham, 383–395. https://doi.org/10.1007/978-3-030-23204-7_32

[19] Thomas W. Price, Samiha Marwan, Michael Winters, and Joseph Jay Williams. 2020. An Evaluation of Data-Driven Programming Hints in a Classroom Setting. In *Artificial Intelligence in Education (Lecture Notes in Computer Science)*, Ig Ibert Bittencourt, Mutlu Cukurova, Kasia Muldner, Rose Luckin, and Eva Millán (Eds.). Springer International Publishing, Cham, 246–251. https://doi.org/10.1007/978-3-030-52240-7_45

[20] Leena Razzaq and Neil T. Heffernan. 2006. Scaffolding vs. Hints in the Assistment System. In *Intelligent Tutoring Systems (Lecture Notes in Computer Science)*, Mitsuru Ikeda, Kevin D. Ashley, and Tak-Wai Chan (Eds.). Springer, Berlin, Heidelberg, 635–644. https://doi.org/10.1007/11774303_63

[21] Kelly Rivers and Kenneth R. Koedinger. 2017. Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education* 27, 1 (March 2017), 37–64. https://doi.org/10.1007/s40593-015-0070-z

[22] Ido Roll, Vincent Aleven, Bruce M. McLaren, and Kenneth R. Koedinger. 2011. Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction* 21, 2 (April 2011), 267–280. https://doi.org/10.1016/j.learninstruc.2010.07.004

[23] Ido Roll, Ryan S. J. d Baker, Vincent Aleven, and Kenneth R. Koedinger. 2014. On the Benefits of Seeking (and Avoiding) Help in Online Problem-Solving Environments. *Journal of the Learning Sciences* 23, 4 (Oct. 2014), 537–560. https://doi.org/10.1080/10508406.2014.883977 Publisher: Routledge _eprint:

https://doi.org/10.1080/10508406.2014.883977.

[24] Benjamin Shih, Kenneth R Koedinger, and Richard Scheines. 2010. Unsupervised discovery of student learning tactics. 201–210.

[25] Benjamin Shih, Kenneth R. Koedinger, and Richard Scheines. 2011. A response time model for bottom-out hints as worked examples. *Handbook of educational data mining* (2011), 201–212.

[26] Teemu Sirkiä. 2018. Jsvee & Kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process* 30, 2 (2018), e1924. https://doi.org/10.1002/smr.1924 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1924.

[27] Kurt Vanlehn. 2006. The behavior of tutoring systems. *International journal of artificial intelligence in education* 16, 3 (2006), 227–265.

[28] Wengran Wang, Yudong Rao, Rui Zhi, Samiha Marwan, Ge Gao, and Thomas W. Price. 2020. Step Tutor: Supporting Students through Step-by-Step Example-Based Feedback. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. Association for Computing Machinery, Trondheim, Norway, 391–397. https://doi.org/10.1145/3341525.3387411

[29] Yeyu Wang, Shimin Kai, and Ryan Shaun Baker. 2020. Early Detection of Wheel-Spinning in ASSISTments. In *Artificial Intelligence in Education (Lecture Notes in Computer Science)*, Ig Ibert Bittencourt, Mutlu Cukurova, Kasia Muldner, Rose Luckin, and Eva Millán (Eds.). Springer International Publishing, Cham, 574–585. https://doi.org/10.1007/978-3-030-52237-7_46

[30] Benjamin Xie, Greg L. Nelson, and Andrew J. Ko. 2018. An Explicit Strategy to Scaffold Novice Program Tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. Association for Computing Machinery, Baltimore, Maryland, USA, 344–349. https://doi.org/10.1145/3159450.3159527