Stepwise Help and Scaffolding for Java Code Tracing Problems With an Interactive Trace Table

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

Jordan Barria-Pineda
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
jab464@pitt.edu

Kamil Akhuseyinoglu
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
kaa108@pitt.edu

Peter Brusilovsky
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
peterb@pitt.edu

ABSTRACT

In this paper, we describe the integration of a step-by-step interactive trace table into an existing practice system for introductory Java programming. These autogenerated trace problems provide help and scaffolding for students who have trouble in solving traditional one-step code tracing problems, accommodating a wider variety of learners. Findings from classroom deployments suggest the scaffolding provided by the trace table is a plausible form of help, most notably increases in performance and persistence and lower task difficulty. Based on usage data, we propose future implications for an adaptive version of the interactive trace table based on learner modeling.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Applied computing** → **Interactive learning environments**.

KEYWORDS

smart learning content, educational tools, tracing, teaching with technology, intelligent tutoring systems, technology integration, help design, educational research, computer science education

ACM Reference Format:
Zak Risha, Jordan Barria-Pineda, Kamil Akhuseyinoglu, and Peter Brusilovsky. 2021. Stepwise Help and Scaffolding for Java Code Tracing Problems With an Interactive Trace Table. In *21st Koli Calling International Conference on Computing Education Research (Koli Calling '21), November 18–21, 2021, Joensuu, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3488042.3490508

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 [35]. Practicing problem-solving skills using this step-by-step approach with ITS support became highly popular in teaching math and physics over the last 20 years [17, 36]. In contrast, systems and tools created in the same period of time to support problem-solving skills in areas of computer science (CS) such as Lisp [38], Java [11, 15], or SQL [6] often expected students to enter the final solution rather than solve the problem step-by-step. One reason for the difference is computer science learning tools of that time predominantly focused on college-level students that needed less scaffolding. However, step-by-step problem-solving support has been recognized as highly valuable for the majority of school-level students. This approach provides a better learning experience for less-prepared students, allowing them to catch errors early and providing scaffolding throughout the process.

Over the last 5 years, the cohort of students taking computer science classes has changed dramatically. The popularity of the field has encouraged a much larger variety of college students to enroll in CS. Programming classes are also becoming more popular in high and middle schools. The need to provide better support for the increasing fraction of less-prepared students in CS classes motivated a number of recent attempts to create ITS supporting step-by-step problem support for CS classes [16, 28, 30, 37]. The step-by-step approach, however, does have a few shortcomings, as having students complete each step can slow down productivity and make the process boring for better-prepared students. 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.

In this paper, we present the design and evaluation of a joint approach that combines the strong sides of the traditional "final answer" approach with ITS-style step-by-step support for practicing program tracing skills in Java. With the joint approach, students start with an opportunity to solve a traditional tracing problem in one step by entering the final solution. However, if a student has trouble in attempting the problem or fails to solve it correctly, he or she could switch to a step-by-step approach. The main challenge of this combined approach is the ability to convert a standard problem with a final answer, which is relatively easy to create, to a step-by-step ITS experience, which usually requires a lot of authoring efforts. To overcome this challenge, we explore an opportunity to automate the authoring process entirely.

Koli Calling '21, November 18–21, 2021, Joensuu, Finland

ACM ISBN 978-1-4503-8488-9/21/11…$15.00
https://doi.org/10.1145/3488042.3490508

2 RELATED WORK

2.1 Step Authoring and Automation

While ITS and their step-by-step problem-solving have often been noted for strong learning gains, a common stumbling block to deployment is the amount of time needed to construct these systems [8, 12]. 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] and ASSISTments [14] 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 problems, thus it is not much surprise that researchers have sought to automate aspects of the creation process. In particular, researchers have tried to rely on particular affordances of a domain to automatically generate steps, hints, and visualizations. O'Rourke [26, 27] 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'Rourke's case, an expert was needed to implement an algorithm that maps to solving a type of mathematical problem, but after was able to consistently generate steps and hints for similar problems within the same domain. Such an approach helps reduce the burden of constructing these systems and provides more varied content.

Code offers many affordances that can make expert mapping unnecessary, allowing for a system to accommodate a broad range of problems without additional modifications. Two notable approaches have been applied to create program visualization tools. JSVEE [34] applied a *transpiler* based on the static analysis of code which produced an output that could be utilized by a visualization tool. This method allows for more detailed evaluation at the expression level and can handle a wide array of problems, but, similar to O'Rourke's approach, requires the manual development of a transpiler to interpret the code. Guo [13] built a visualization tool, the Python Tutor, but instead relied on *execution traces* produced by debugging instruments, often found in programming languages, to generate 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.

2.2 Code Tracing

While the affordances of code have been used to create visualizations, they can also be used to generate problems for a more interactive user experience. The *execution traces* produced by debuggers contain sufficient information to break a problem into steps. In contrast to some earlier attempts to implement step-by-step code tracing [5, 19], 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 popular strategy to practice code tracing skills. It involves tracking state changes, such as variable values, throughout a program's execution.

Paper-based trace studies can draw their origins from a multinational study of student code reading skills conducted by the Leeds Working Group at ITiCSE 2004. Part of this study involved analyzing sketches participants made on scratch paper and test sheets. The analysis revealed sketching traces to be correlated with higher success on code reading problems involving loops, arrays, and conditionals. More recently, studies have replicated this finding [9], noting a higher success rate of students who sketched out traces or applied an explicit line-by-line tracing strategy [39]. Studies also noted correlations between code tracing and code writing skills [18, 19, 21], suggesting transfer to more advanced tasks.

In modern practice, researchers exploring the use of trace tables in education [9, 10, 39] frequently contrast their work with interactive visualization tools like JSVEE or Python Tutor [13, 34], referring to them as passive and lacking active engagement a trace table provides. In this context, our work attempts to bring together the best of both approaches: the familiar and efficient format of trace tables and the automatic trace construction provided by the visualization tools. Our software version alleviates the burden of learning or applying tracing on paper, while also allowing integration into learning systems to collect fine-grained learner data.

2.3 Help and Scaffolding in Tutoring Systems

Help can come in many forms in tutoring systems and paper-based tracing has often been situated as an aid for solving other coding problems. Prior research on integrating help in tutoring systems has reported many challenges, sparking investigations into 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]. Likewise, there has been significant research in the automation of this help, such as hints, for each sequential step of a problem based on prior student data [4, 23, 30]. Recent research has gone a step further and studied the impact of such automated hints in the context of block-based programming with mixed results, either noting performance gains but requiring additional self-explanations [22] or having no impact on performance or learning [28]. As demonstrated, 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].

Some studies have questioned the degree to which hints are useful as a form of help, noting that reattempting a problem, rather than obtaining a hint, may be better for learning [32]. This bring to the forefront the question of *how* a system should offer help. Researchers have noted the potential of replacing or augmenting hints with more interactive activities such as self-explanations or scaffolding [2, 7, 22, 29]. Such an approach was adopted for math in ASSISTments, which compared manually breaking a single step problem into either a sequence of hints or a step-by-step problem [33]. Results noted higher learning gains for the step-by-step condition, encouraging future research using automated methods in a different domains such as programming. An active scaffolding strategy like the trace table may be more suitable and easier to automate than direct help approaches like hints.

3 THE SYSTEM

To combine traditional "final answer" problems with an ITS-style step-by-step support, we developed a tool that automatically generates an interactive trace table problem from a given piece of source code. With this tool, most "final answer" problems could

[1] http://pythontutor.com/

be converted into an step-by-step experience. The trace table interface provided by the tool allows students to navigate through the program's execution and provides immediate feedback of the correct change in a given step. Students are actively engaged as they track and predict the execution of a program, providing automatically constructed scaffolding for students that helps them diagnose errors. Our interface (Figure 1) draws inspiration from paper-based methods described in prior research [9], coming closest in resemblance to a 'line' sketch trace table where variables are represented in columns. At the top, users see the code being traced, highlighting the current line being executed. Both 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.

While auto-generated trace table problems could have many pedagogical applications, we investigate its use as a form of help by integrating it into an existing system, QuizJet [15], as a supplementary activity to provide learner support (Figure 2). QuizJet offers traditional "final answer" problems where users must respond with the final value or output of a program. These problems do not utilize a step-by-step approach and consist of a single observable step. QuizJet features parameterized problems which are problems that are instantiated with randomly generated values upon every access. Upon failure, students are provided the correct answer but the problem parameters will be altered in future attempts. In essence, this creates a different version of the problem with a different answer.

In the context of help and scaffolding in tutoring systems, our attempt-based design actively engages students through the process. In lieu of giving a hint, the system provides a remedial 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 the 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

Figure 1: The Trace Table interface

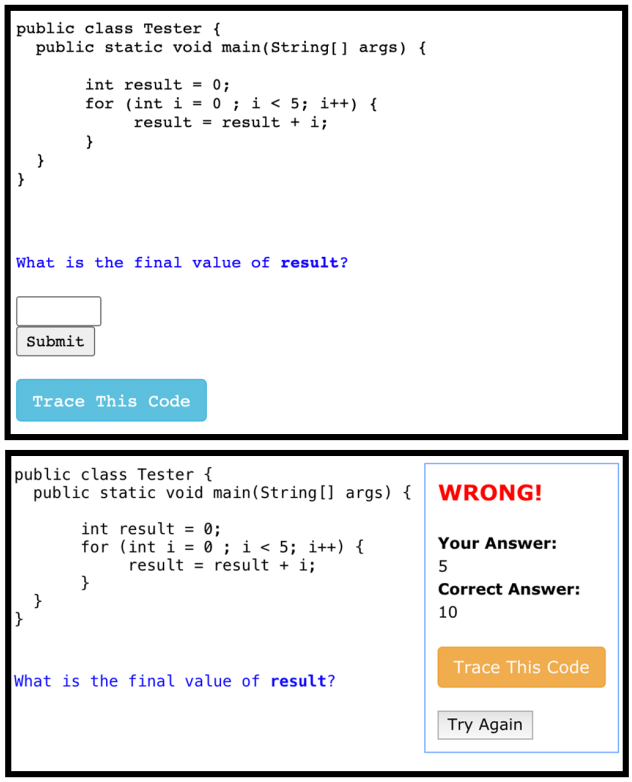

Figure 2: Trace table access before (top) or after (bottom)

support that the trace table can provide. Therefore, this form of help actively scaffolds students in tracing as opposed to providing a principle-based hint. By using the trace table, students can identify the specific steps where they have made a wrong assumption and also better track the execution of the program. Likewise, the broken down tasks also help the system diagnose areas where a student may be deficient. As shown in Figure 2, 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.

Since this form of help is provided on-demand, and to address the issue of when students should engage with the tool, we implement a popup recommendation (Figure 3) in our experiment group based on a student model. The trace table is recommended after students fail on newer problems they have less experience with. This is inferred through the system's learner model [40] which calculates student knowledge for each concept associated with a problem. The system averages the concept knowledge for that problem and if below a threshold, we trigger the popup after an incorrect attempt. This is not only an effort to combat help avoidance but also ensure students are familiar with the help at their disposal.

The trace table interface was implemented 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 [13]. To address learner modeling

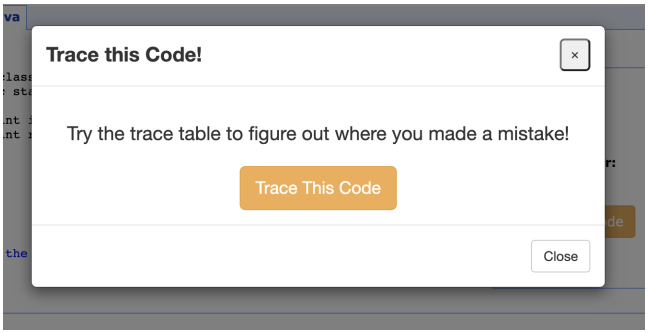

Figure 3: A popup recommendation conditionally appearing to users after failing

needs, knowledge components are mapped to each line automatically using a java parser which utilizes the abstract syntax tree to parse the code [20]. 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 the inclusion of a JavaScript and CSS file and provides an API to render and configure the trace table on a web page.

4 FIRST-ROUND STUDIES

4.1 Practice System Study

A practice system study of the trace table tool was conducted over two semesters in IS100, an undergraduate introductory Java programming course. Subjects in this study were students enrolled in either one of two sections in Fall 2019 or one section in Spring 2020. Students in the course were not required to have prior programming experience. The trace table tool was an optional feature made available to all students as part of the practice system to use at their own discretion. The practice system included several types of interactive learning content, including quizzes via the QuizJet system. All problems were accessed through the Mastery Grids system [20] 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.

Prior to the introduction to the system, students completed a 12 question pretest at the start of the semester consisting of Java trace problems similar in nature to those in the QuizJet system. Students were randomly assigned to a popup or no-popup group that determined whether the system would recommend the trace table via a popup or no popup would ever be shown. Study conditions were the same between Fall and Spring implementations with one exception. In the Spring, an additional button was added (Figure 2) to QuizJet questions to allow users to trace before submitting an answer. To incentivize students to use the practice system, up to a 2% increase of their final grade was offered as extra credit for solving at least one problem for each of the 19 topics, 12 of which were supported by the trace table tool. Over 101 unique QuizJet problems were included in the system, 66 of which included a trace table option.

Enrollment from the first round as well as all other deployments is listed in Table 1. The first round consisted of 77 students enrolled in the three sections of IS100. Of the 50 users that accessed the system, 26 were part of the popup group and 24 were part of the no-popup group. Across all 3 sections, 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. For spring, the new option to trace before an attempt was used for 24.6% of trace tables. Our research questions for this practice system study were:

- Would automatically breaking a single-step trace problem into multiple steps improve performance and serve as an adequate form of help?
- Does use of the trace table impact other behavior in the system, such as increased persistence or overall usage?
- Would a popup recommendation based on the student model successfully combat help avoidance and promote use of the trace table?

4.2 Recitation Study

In addition to the practice system study that investigated long-term use of the trace table tool, we conducted another study during Fall 2019 to look at short-term effects and the impact of tracing more advanced problems. This study was performed as a part of a recitation lab in a CS100, a similar introductory Java course, during the last two weeks of the semester. The recitation was focused on code tracing skills and was supported by the QuizJet system. During the recitation, students were expected to solve 8 QuizJet problems. In contrast to the practice system context, the lab version of QuizJet did not show a correct answer upon incorrect attempts. Optional access to the trace table was also provided before an attempt to a QuizJet problem which was used for 30.6% of attempts. The students could complete the whole set either in a lab setting or at students' leisure at home. The recitation was offered as a way to prepare for the upcoming course final, however, participation was optional. A total of 90 users participated in the recitation study 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. For this single session application, we were interested in the additional question:

- Would the trace table tool provide adequate help and support in a short-term application with more advanced problems?

5 FIRST-ROUND RESULTS

We performed a variety of analyses using log data collected in both practice and recitation settings. We incorporated the recitation data in our analysis of Improving Performance (5.1), Help After Failing (5.2) and Other Observed Usage Behavior (5.5), while the remaining analysis comes only from practice system data.

Table 1: Study Enrollment and System Usage Stats

Course Name	Semester	Total Students	Used System	Used Trace	QuizJet Attempts	Trace Attempts
IS100 A	Fall '19	14	8	6	704	58
IS100 B	Fall '19	30	16	15	1910	109
IS100	Spring '20	33	26	14	2366	130
CS100*	Fall '19	146	90	80	1682	447
ISCS400 A	Fall '20	39	15	15	449	72
ISCS400 B	Fall '20	66	38	20	839	84
Total	-	328	193	150	7950	900

*CS100 was a short term recitation deployment rather than a practice system deployment

5.1 Improving Performance

Our analysis began by investigating the trace table tool's ability to improve performance of students which could decrease frustration and provide more opportunities to reinforce knowledge. Initially we compared difficulty which was inferred by student error rate (any incorrect response to a QuizJet problem or trace table step). To eliminate some of the noise in the data from easy problems, we looked at the tougher half of problems based on difficulty, i.e., the mean error rate. The mean error rate of completed trace table problems was lower at 18.1% with over 47.2% of the tables reporting 100% accuracy as opposed to QuizJet with 35.9% mean error rate and only 39.7% with no errors (Figure 4). To better understand engagement, we calculated the average time students spent on each step of a traced table. Each trace table contains multiple steps and some steps are fairly simple (e.g. variable initialization). To account for disengagement, we considered the 95th percentile of data based on total duration of the problem. Users in the practice system spent an average of 7.6 seconds (SD = 5.3) on a step.

In the recitation context, students displayed lower levels of accuracy. As opposed to the practice system context, the recitation took place in a condensed time frame and used more challenging problems. In comparison with the practice system study, the mean error rate increased for the trace table (35.7%) and QuizJet (48.9%). Instances of no errors were rarer for the trace table (15.9%) and QuizJet (30.4%). Users in the recitation study spent an average of 12.6 seconds (SD = 8.2) on a step (again using 95th percentile based on total problem duration). While the trace table reported lower levels of error, the different structure of activities makes direct comparison less reliable.

For a more reliable alternative, we analyzed how students performed with each respective activity over time as they practiced and their number of learning opportunities (i.e., attempts of activity type) increased. Our analysis focused on the practice system study which spanned an entire semester. For analyzing performance over time, we used a mixed effects logistic regression using the binary outcome of the attempt (correct or incorrect) as our dependent variable. For our independent variables, we used the opportunity number of each activity (QuizJet or trace table) as a fixed independent variable, and the student and problem as random independent variables. A mixed effects model was selected to handle uneven repeated measures of users and problems in our data. Additionally, a log transformation was used on opportunity number in order to fit the model so it converges and to capture relative change. For QuizJet, opportunity number was not found to be a significant predictor of outcome $\beta = -0.10$, $z(4979) = -1.4$, $p = .157$. However, for the trace table, opportunity number was a significant predictor of outcome $\beta = 0.43$, $z(2439) = 5.4$, $p < .001$. These results suggest that while increased practice and familiarity had little effect for QuizJet performance, they did significantly increase the success of students for the trace table.

5.2 Help After Failing

We also wanted to investigate whether the trace table aided students in solving QuizJet problems they had previously failed on. As explained above, QuizJet problems were parameterized so students could try the same problem again and again. A sequence of incorrect answers could lead to frustration or shallow learning. For the practice system study, the only other form of help was feedback of the correct answer to an incorrect attempt. The recitation study only provided feedback of whether an attempt was correct or incorrect. We looked at instances when users failed and reattempted the problem without tracing compared to instances when users failed, completed a trace table, then reattempted the problem.

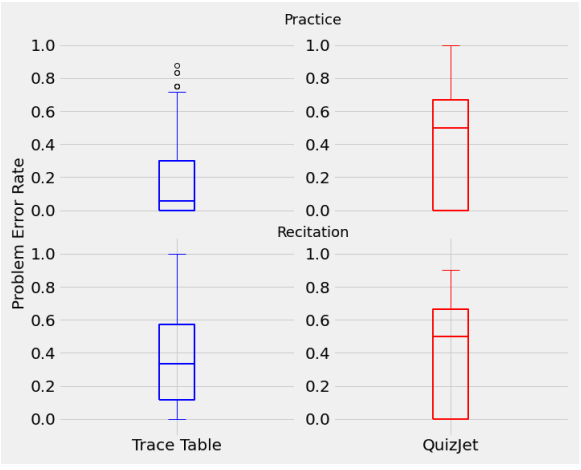

Figure 4: Comparing QuizJet and Trace Table error rate in practice system and recitation studies

A mixed logistic regression was used again to examine the effect between a user completing a trace table and answering correctly on the reattempt. For the practice system study, results indicated no significant effect between completing a trace table and answering correctly $\beta = 0.27$, $z(670) = 0.9$, $p = .385$. However, we conducted the same analysis for the recitation with a significant, and strong effect $\beta = 1.36$, $z(1061) = 5.2$, $p =< .001$, $OR = 3.91$. Students were 3.91 times more likely to answer the problem correctly after tracing, suggesting the trace table was an adequate form of help even in this single session. However the practice system did not yield the same results, suggesting that other factors might limit the effectiveness of the tool.

5.3 Persistence in the System

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. If after a sequence of attempts the student can finally solve the problem correctly, it becomes a valuable learning opportunity. However, if a sequence of attempts ends with a wrong answer, the chance to learn will be low. Abandonment, failure to solve an attempted problem, was infrequent, occurring only 30 times total in the practice system 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, could impact behavior and contributed to students persisting and solving the problem. However, we did not find any evidence suggesting that using the trace table was associated with using the system more (i.e. solving more QuizJet problems).

5.4 Usage and Recommendations

Not all students made use of the trace table with 35 of 50 completing at least one problem, and 15 of those 35 completing 4 or more tables. Prior research has noted help avoidance and worked to remedy it [3, 31] and, more recently, reluctance to trace on paper [10]. To account for this, we attempted to promote the use of the trace table for struggling students. Students randomly assigned to the pop-up group would have a popup 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 practice system study, the mean conversion rate (percentage tracing upon recommendation) of users was 65%, indicating that the recommendation to trace was fairly successful based on the model. Roughly the same number of users made use of the tool in each condition, 69.2% in the popup recommendation group and 62.3% in the no-popup group. We looked further at the 35 students that made use of the tool, comparing the number completed between each group. A Mann-Whitney U test comparing the trace tables completed by the popup recommendation group ($Mdn = 8$) with the no-popup group ($Mdn = 3$) was marginally significant $U(50) = 187, n_1 = 19, n_2 = 16, p = .06$. This evidence could suggest the recommendation promoted higher use of help, but not necessarily whether help was ever used.

5.5 Other Observed Usage Behavior

There was little evidence of "gaming" the system, like the "bottoming out" of hints, which might be attributed to the parameterized nature of the problems making gaming less effective. The few instances of gaming observed were actually due to problems with an excessive number of steps. One such case involved a nested loop which generated over 150 steps for the student to complete. Other similar idiosyncrasies were observed, such as input errors over the specificity of data types (e.g., including a decimal and trailing zero for floats and doubles). 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 rarer in the practice system study where usage occurred over a period of time and error rates were lower.

Our analysis also sought to account for the time investment of the trace table. If the trace table significantly increased the time needed to solve problems, it could be a less efficient form of help. For our analysis, we selected instances where users opened and first solved a QuizJet problem, calculating the total time elapsed. After, instances were separated based on whether or not a trace was completed within that span of time. To account for excessive times from disengagement, we took the 95th percentile of data based on total duration. We grouped the data by the 29 users that traced prior to their first correct answer of a problem and calculated the mean duration for traced and non-traced instances. A Wilcoxon signed-rank test found that mean duration of users' traced instances ($M = 47.7$, $SD = 39.8$) was not significantly different from non-traced instances ($M = 37.9$, $SD = 10.8$) $Z = 192$, $p = .58$. This suggests that when not tracing, students still consumed comparable amounts of time due to other factors (e.g. additional errors or time thinking).

6 SECOND-ROUND STUDY

While our pilot study suggested the trace table was helpful for performance and provided better learning opportunities, it also revealed a lower than expected engagement with many users who opted to simply reattempt problems rather than trace. Likewise, we believed the tool might have less value to better-prepared students. Our subsequent work sought to increase the value and appeal of the trace table technology to a more diverse range of users.

6.1 Study Design

Our study followed a similar design to the first-round practice system study, although only a single condition was used as all users received popup recommendations based on the student model. As the previous course IS100 was discontinued, the system was deployed in the replacement intermediate course ISCS 400, which contained similar curriculum with some additional content on algorithmic complexity and added emphasis on recursion. The more notable difference was that both computer science and information science majors could enroll. There exists approximately 3 times as many CS students in the department and they are required to have a prior course in programming as opposed to information science

Figure 5: A trace table in worked example mode (left) and various options for users after failing (right)

majors. Exact distribution of major was difficult to determine as many students were undeclared freshmen. There is no evidence to suggest the sample student proportions differed from the population with regard to major. The change in course prompted an analysis of pretest scores to determine if the population characteristics had changed. Pretest scores (Figure 6) of the 105 students in the second-round practice system study (M = 6.86, SD = 2.81) compared to the 75 students in the first-round practice system study (M = 3.57, SD = 2.58) were significantly higher, $t(178) = 8.0$, $p < .001$. Similarly, we compared pretest scores for only those that accessed the practice system. The 52 students in the second-round study (M = 7.37, SD = 2.59) compared to the 51 of the first-round study (M = 3.57, SD = 2.33) were still significantly higher, $t(101) = 7.8$, $p < .001$. These observed differences presented a new opportunity to engage a more diverse set of users. In addition, subjects in our first-round study did not differ dramatically in skill, but combined with our second deployment we ensured the trace table was tested on a wide range of prior knowledge and preparation.

For increasing the trace table's appeal to users, including those better prepared, we developed two additional trace modes. These two modes introduce a new form of interaction, allowing students to click next through the trace, rather than requiring an input (Figure 5). The entire set of help options were as follows:

- Trace All Code: A trace table where users are prompted to input all variable changes.
- View & Trace Code: A trace table where only some steps require user input.
- View Code Trace: A complete worked example where no user input is required.
- View Answer: Show the answer to the problem.

Our new modes served as forms of worked-examples, allowing users to see the code trace demonstrated step by step. To promote the trace table and encourage more deliberate use of help for users after failing (shown in Figure 2), we removed the default behavior of showing the correct answer after failing. Users could still see the answer but were required to click the corresponding button. Our research questions were:

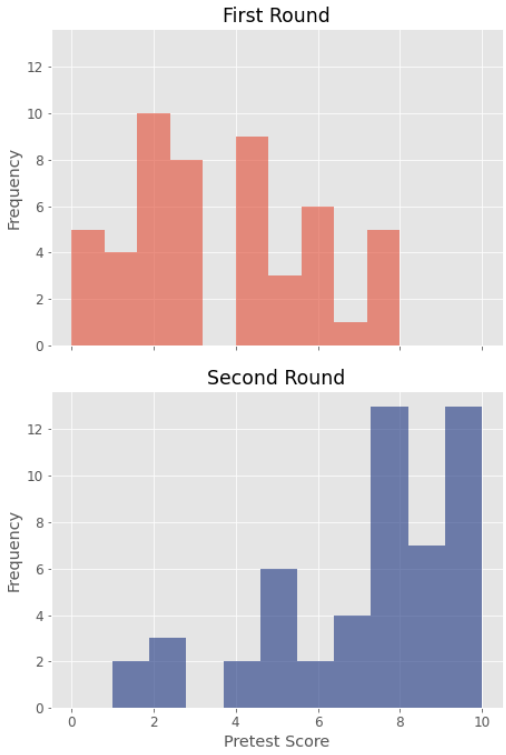

Figure 6: Histogram plot comparing pretest scores of first-round (top) and second-round (bottom) studies

- Would adding additional modes to the trace table increase its appeal to better-prepared students?
- How would the three trace modes differ in appeal and usage by students in the practice system?

Between two sections, 105 students participated in the pretest and 53 of which accessed the practice system (Table 1). Of these 53, 35 used a trace-based method of help, the lower number likely due to limited system use. There were a total of 1288 QuizJet submissions

to 51 unique problems that were supported by the trace table. This resulted in a mean of 24.3 QuizJet submissions per user, in contrast with 99.6 in the first-round practice system study.

6.2 Second-Round Study Results

There was notably less overall system usage by students; however, the trace tool usage seemed to proportionately increase. To clarify, the initial practice system study, there were 4980 QuizJet attempts and 297 trace tables for a ratio around 17:1. However, the second-round study had only 1288 QuizJet and 156 trace table attempts for a ratio around 8:1. Despite having nearly one-fourth of the QuizJet submissions, the second-round study produced proportionately twice as many trace attempts. While popup recommendation was enabled for all of the second-round study but half of the first, we still see a similar ratio when removing those accessed via a popup. However, completion rates did differ as the first-round trace tables were completed 86.2% of the time which dropped to 55% in the second round. For the second round, we further looked at partial completion (over half steps completed) which was higher at 68.6% and could indicate obtaining partial knowledge from the exercise. While the additional trace modes seemed to increase the appeal of the tool, they were not always sufficient for sustaining engagement.

Although overall usage of the trace table increased, many users still chose other options after failing. Despite no longer providing a correct answer, the majority of users elected to immediately retry the problem. The average among users for retrying after failure was 54.3% of the time. The next highest choice was to show the answer at 27.5% of the time, followed by tracing (any of the three options) at 18.3%. In an effort to account for prior knowledge, we tried to correlate user choices with pretest scores but were unable to find any significant results. Still, these rates suggest a degree of avoidance evident in the high percentage of students reattempting problems or just seeking the correct answer.

With regards to the specific trace options, students tended to select those that required less effort and cognitive load. Among all users that traced, the worked example was the most commonly selected option with a mean of 43.8% followed by 28.6% for worked-example with partial tracing, and 27.6% for a full trace. These findings, along with the higher rates of choosing to show the answer, suggest a preference for easier and quicker forms of help. To further analyze the impact of time, we calculated the mean duration per step of each traced table and grouped it by each of the three different modes (Figure 7). We took the 95th percentile of duration to account for excessive times from disengaged users. As expected, the worked-example mode was much faster, but the low mean of 2.47 seconds could indicate a shallow engagement with the activity. Unfortunately, we did not collect sufficient data to analyze the differing impacts on performance among modes.

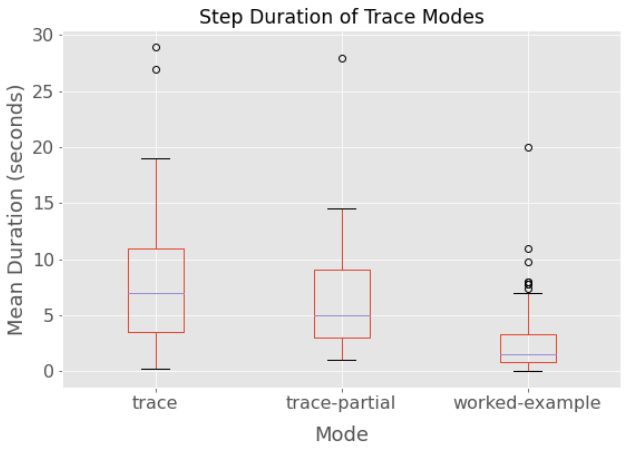

Figure 7: Comparing mean step duration among all three trace modes

7 DISCUSSION

7.1 Evaluating Help Efficacy

We expected that breaking problems down using the trace table would improve performance and serve as adequate help, yet our results were not always consistent. In the practice system study, student accuracy improved over time with more use of the tracing tool (as opposed to QuizJet). This was a positive finding, as the trace table tool created more opportunities to reinforce knowledge and helped reduce difficulty, which could lead to better learning opportunities for students. Likewise, there was some evidence that the trace table helped students persist on challenging problems. However, when looking at the tool's ability to help after failure, there was no observed difference when using the tool in the practice system study. This finding differed in the recitation study where learners did show improvement after tracing, providing evidence that the trace table was indeed helpful for students in solving problems they had failed on. This result even occurred despite the tool being used in a short-term application. These conflicting findings between deployments imply that other factors might affect the tool's ability to serve as an adequate form of help. For instance, the recitation used more advanced problems and was aimed at preparing for a final, suggesting that the complexity of a problem or motivation of students could impact the tools efficacy. Additionally, other forms of help may have already been sufficient for simpler problems, as students were provided feedback of a correct answer in the practice system context. Last, another factor could be the level of engagement, as students spent more time on average tracing in the recitation. Overall, trace table's ability to serve as an adequate form of help seemed to depends on a number of factors like motivation, engagement, or problem complexity.

7.2 Help Avoidance and Appealing to Users

Our concerns regarding help avoidance seemed justified based on usage in our deployments. Results of the initial effort to limit help avoidance using a popup recommendation based on a student model suggested that such an approach could increase usage. However, there was little difference for whether users tried the tool or not. Overall, a popup recommendation might help, but was not a sufficient solution on its own. Similarly, help avoidance was a concern for our second-round study. Adding in additional modes of tracing resulted in a higher ratio of usage, producing proportionately twice as many trace attempts. These additional modes seemed to increase

the appeal of the trace table, even with a more prepared population, however most users still elected to retry a problem or view the correct answer when struggling. Students favored similarly quicker options for the trace table, such as the worked-example mode. This usage could reflect a tendency to select options that require less effort or time, rather than those more beneficial for learning. Such observations could also be explained by high pretest scores of students in the second-round study who may have found the higher level of scaffolding unnecessary. Beyond the reason for avoidance, these findings bring forefront the need to maintain engagement of learners. Our second-round study also observed a notable abandonment of traces or partial completions. This lack of engagement could be related to students not requiring all the information provided in a trace or growing bored by the activity. For example, some types of problems result in a time-consuming intervention that could impede learning. Loops in general produce repetitive tasks in tracing which are sometimes useful but often unnecessary. Frustrating or tedious experiences could result in users avoiding the form of help for future issues, highlighting the need to customize aid and minimize unnecessary burdens.

7.3 Intelligent and Adaptive Recommendations

These issues related to help efficacy, engagement, and avoidance all call for more intelligent and adaptive solutions that can preserve the autonomous nature of the trace table yet tailor it to each learner. Automatically identifying what code is most suitable for applying this strategy could increase efficacy. Likewise, the tool could track users' behavior, such as accuracy and time spent, to ensure learners are using the tool effectively. Additionally, the observed preferences for faster solutions in our deployment calls for adaptations that shorten the task by eliminating easier or repetitive steps based on the model of 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. Such changes could improve engagement, not only aiding the efficacy but also increasing appeal.

Identifying the optimal moments for recommendation could further reduce avoidance. Modeling user motivation and knowledge could identify when this form of help will make an impact. Additionally, an adaptive trace table could specify when to offer or recommend specific modes of help based on the student's knowledge and prior behavior. Adaptively offering tracing before an attempt could aid students that need more initial support. Likewise, selecting a mode like worked-example could lower cognitive load for less experienced students. Such an adaptive design would be similar to step-based help recently proposed in program construction tasks [37]. Additionally, interleaving worked examples and problems might make students more efficient and decrease student frustration as observed in prior research [24, 25]. Overall, these modifications would ensure the trace table is introduced when needed most and help balance the extra investment of time. An intelligent approach could foster trust with learners that the help provided will match their needs.

7.4 Limitations

Our second-round study noted a decrease in participation from that observed in the first-round study. While potentially explained by differences in pretest scores, there also might be alternative explanations, such as digital burnout from remote education ushered in by COVID-19. Students may have found additional screen time in the practice system beyond their online courses tedious.

Additionally, participants in this study were provided both the practice system and trace table as optional learning aids. This dynamic introduces self-selection bias as learners can choose whether or not to participate in using these tools. This limitation means that the recruited sample might differ from the target population in characteristics like motivation. Therefore, some conclusions from this study might not generalize to all groups of learners.

While this study noted some improvements in student performance, further analysis needs to be conducted to investigate the trace table's relationship with learning gains of students. Unfortunately due to the disorder from COVID-19, we were unable to consistently collect and conduct the posttest for a reliable analysis of learning gains. Similarly, the trace table was only compared with a simple form of help (providing the correct answer to incorrect responses). Therefore, we cannot conclude the tool would outperform other automatically generated forms of help.

8 CONCLUSION

This study investigated the use of an automatically generated trace table as a help mechanism for students. Analysis of log data suggests that automatically breaking down the task into steps reduced error and produced a better learning opportunity for students. Data suggested students' accuracy did not change with repeated use of QuizJet, but more practice with the trace table did have a positive effect on accuracy. Additionally, there was strong evidence the trace table aided in the recitation study by improving the accuracy of students on failed problems. Both these findings build on prior recommendations and results in support of more interactive methods of help like step-by-step scaffolding. Similarly, our analysis suggested that the trace table provided enough support to help students persist through difficult problems despite the active, attempt-based nature of the exercise. Additional modes of the trace table that converted the activity to a worked-example increased its appeal to better-prepared learners while also reducing time on task. Adaptions based on learner modeling could increase engagement and efficacy by recommending appropriate modes and optimizing the amount of work of the trace table.

ACKNOWLEDGMENTS

This material is based upon work supported by University of Pittsburgh through Innovation in Education Award and the National Science Foundation under Grant No 1822752.

REFERENCES

[1] Vincent Aleven, Bruce M McLaren, Jonathan Sewall, and Kenneth R Koedinger. 2006. The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In *International Conference on Intelligent Tutoring Systems*. Springer, 61–70.

[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

(2016), 205–223.
[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.
[4] Tiffany Barnes and John Stamper. 2008. Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems*. 373–382.
[5] Peter Brusilovsky and Tomasz D Loboda. 2006. WADEIn II: A case for adaptive explanatory visualization. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*. 48–52.
[6] Peter Brusilovsky, Sergey Sosnovsky, Michael V. Yudelson, Danielle H. Lee, Vladimir Zadorozhny, and Xin Zhou. 2010. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *ACM Transactions on Computing Education* 9, 4 (2010), 19:1–19:15.
[7] 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.
[8] Albert T. Corbett, Kenneth R. Koedinger, and John R. Anderson. 1997. Intelligent Tutoring Systems. In *Handbook of Human-Computer Interaction*. North-Holland, 849–874.
[9] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. 2017. Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. 164–172.
[10] Kathryn Cunningham, Shannon Ke, Mark Guzdial, and Barbara Ericson. 2019. Novice Rationales for Sketching and Tracing, and How They Try to Avoid It. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. 37–43.
[11] Stephen H. Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. 188–193.
[12] Jeremiah T Folsom-Kovarik, Sae Schatz, and Denise Nicholson. 2010. Plan ahead: Pricing ITS learner models. In *Proceedings of the 19th Behavior Representation in Modeling & Simulation (BRIMS) Conference*. 47–54.
[13] 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)*. 579–584.
[14] Neil T. Heffernan and Cristina Lindquist Heffernan. 2014. The ASSISTments Ecosystem: Building a Platform that Brings Scientists and Teachers Together for Minimally Invasive Research on Human Learning and Teaching. *International Journal of Artificial Intelligence in Education* 24, 4 (Dec. 2014), 470–497.
[15] 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.
[16] Yun Huang. 2018. *Learner Modeling for Integration Skills in Programming*. Ph.D. Dissertation.
[17] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. 1997. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education* 8 (1997), 30–43.
[18] Amruth N. Kumar. 2013. A Study of the Influence of Code-Tracing Problems on Code-Writing Skills. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. 183–188.
[19] Amruth N. Kumar. 2015. Solving Code-tracing Problems and its Effect on Code-writing Skills Pertaining to Program Semantics. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*. 314–319.
[20] Tomasz D. Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. 2014. Mastery Grids: An Open Source Social Educational Progress Visualization. In *European Conference on Technology Enhanced Learning*. 235–248.
[21] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between Reading, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the Fourth international Workshop on Computing Education Research (ICER '08)*. 101–112.
[22] 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, 61–70.
[23] 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. In *The Proceedings of the Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes (ITS '04)*.
[24] 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.
[25] 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.
[26] 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, 1545–1554.
[27] Eleanor O'Rourke, Eric Butler, Armando Díaz Tolentino, and Zoran Popović. 2019. Automatic Generation of Problems and Explanations for an Intelligent Algebra Tutor. In *20th International Conference on Artificial Intelligence in Education (AIED '19)*. 383–395.
[28] Thomas W Price, Samiha Marwan, Michael Winters, and Joseph Jay Williams. 2020. An Evaluation of Data-Driven Programming Hints in a Classroom Setting. In *21st International Conference on Artificial Intelligence in Education (AIED '20)*. 246–251.
[29] Leena Razzaq and Neil T Heffernan. 2006. Scaffolding vs. Hints in the Assistment System. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*. Springer, 635–644.
[30] 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.
[31] 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.
[32] Benjamin Shih, Kenneth R Koedinger, and Richard Scheines. 2010. Unsupervised Discovery of Student Learning Tactics. In *Proceedings of the 3rd International Conference on Educational Data Mining*. 201–210.
[33] Prawal Shrestha, Ashish Maharjan, Xing Wei, Leena Razzaq, Neil T. Heffernan, and Cristina Heffernan. 2009. Are Worked Examples an Effective Feedback Mechanism During Problem Solving. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*. 1294–1299.
[34] Teemu Sirkiä. 2018. Jsvee & Kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process* 30, 2 (2018), e1924.
[35] Kurt Vanlehn. 2006. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education* 16, 3 (2006), 227–265.
[36] Kurt VanLehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. 2005. The Andes Physics Tutoring System: Lessons Learned. *International Journal of Artificial Intelligence and Education* 15, 3 (2005), 147–204.
[37] 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)*. 391–397.
[38] Gerhard Weber and Peter Brusilovsky. 2001. ELM-ART: An Adaptive Versatile System for Web-based Instruction. *International Journal of Artificial Intelligence in Education* 12, 4 (2001), 351–384.
[39] Benjamin Xie, Greg L. Nelson, and Amy 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)*. 344–349.
[40] Michael Yudelson, Peter Brusilovsky, and Vladimir Zadorozhny. 2007. A User Modeling Server for Contemporary Adaptive Hypermedia: An Evaluation of the Push Approach to Evidence Propagation. In *User Modeling 2007*, Cristina Conati, Kathleen McCoy, and Georgios Paliouras (Eds.). Springer, 27–36.