Giving Hints Is Complicated: Understanding the Challenges of an Automated Hint System Based on Frequent Wrong Answers

Kristin Stephens-Martinez
Duke University
Durham, NC, U.S.A.
ksm@cs.duke.edu

Armando Fox
UC Berkeley
Berkeley, CA, U.S.A.
fox@cs.berkeley.edu

ABSTRACT

Formative feedback is important for learning. Code-tracing is a vital skill in computer science learning. We set out to deliver formative feedback to students on code-tracing, constructed-response assessments by building a student error model using insights gained from inspecting the assessment's frequent wrong answers. Moreover, we compared two different kinds of hints: reteaching and knowledge integration. We found wrong answer co-occurrence provides useful information for our model. However, we were unable to find evidence in our intervention experiment that our hints improved student outcomes on post-test questions. Therefore, we also report here our results on a retrospective, exploratory analysis to understand potential reasons why our results are null.

CCS CONCEPTS

• **Social and professional topics → CS1**; *Student assessment*; • **Applied computing → Computer-assisted instruction**;

KEYWORDS

hints, code-tracing, wrong answers, constructed-response

ACM Reference Format:
Kristin Stephens-Martinez and Armando Fox. 2018. Giving Hints Is Complicated: Understanding the Challenges of an Automated Hint System Based on Frequent Wrong Answers. In *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18)*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3197091.3197102

1 INTRODUCTION

Formative feedback is important. Specifically, we mean "information communicated to the learner that is intended to modify his or her thinking or behavior to improve learning" [16]. In computer science, code-tracing is an important, and potentially necessary, skill that supports code writing [12, 13, 20]. Therefore, we set out to improve student's code-tracing skills by delivering formative feedback given as hints to students using the data from Stephens-Martinez et al. [18] – here after called the TAGS data – who inspected frequent wrong answers from code-tracing questions. However, there is little work examining what kind of hints best helps with learning code-tracing. Therefore, we also investigated the effectiveness of two kinds of hints: reteaching and knowledge integration (KI) [6].

In this work, we report on the process and results of conducting an in situ experiment with 1,057 students, answering 9 question sets, during a 15-week course. With the TAGS data, where a tag represents a student difficulty, we found co-occurrence between wrong answers yields useful information about machine-marked-wrong answers without having to inspect them. With the tags and this insight, we built a student error model. We then used the model to deliver hints to three treatment groups: control (no hints), reteaching, and KI. However, our experiment provided insufficient evidence that giving hints to students with this model improves performance on post-test questions.

Therefore, we performed a retrospective analysis to explore why our results are null. We found undesirable behavior in our system and our students, including Human-Computer Interaction (HCI) problems, students brute-force guessing, and the model delivering too many hints. Moreover, we considered the percent of times where there were zero, one, or more wrong answers after a hint or wrong answer from a set. We found more areas for further investigation, such as students submitting fewer wrong answers after a frequent wrong answer compared to an infrequent one, reteaching hints may help students more than KI hints, and the hint "Typo?" on conceptually correct wrong answers may also help students.

2 RELATED WORK

Le et al.'s [11] review of AI-supported tutoring for learning programming noted that most computer science (CS) tutors give feedback on code writing. Moreover, Le et al. [11] and Holt et al. [9], report that the student model used by these tutors focus on creating an ideal student knowledge model and then modify this model, often by a bug library, to match the student's behavior. Once this behavior is matched, the identified modification(s) is used to generate feedback to the student. Our work does not focus on programming, instead our assessments are code-tracing where students predict the output of code. In addition, our student model is built using hand labeled common wrong answers, which covers a majority of students' wrong answers [18], without any ideal knowledge model.

Moreover, we used data from and we conducted our experiment with constructed-response assessments. The closest related work we could find that gave feedback on code-tracing assessments used multiple choice assessments [4, 10]. Baffes and Mooney's [4] tool ASSERT used theory refinement to automatically alter a student model to become consistent with a set of examples of a student's behavior. The tool assessed the students on C++ code-tracing using multiple-choice questions that asked students to classify the behavior of a piece of code, such as if it would error out. The feedback had two parts. One was a reteaching message that explained the knowledge altered in the student model to fit the student's answer. The second part was a counterexample pertinent to that knowledge. Kumar's [10] code-tracing feedback tool also used C++ code and assessed student understanding through multiple-choice questions. This tool allowed for assessing students with multi-stage, multiple-choice questions such that subsequent questions are dependent on prior ones. When a student is incorrect, a step-by-step explanation of the code is automatically generated and presented to the student.

Finally, our work compares two different kinds of hints, which related work has not done despite the broad literature on formative feedback in education research [7, 14–16].

3 DATA

We have two data sets that came from the same automatic, question-answer system, called OK [5]. One data set is the TAGS data. The other was collected during our experiment. OK administers questions in *question sets* that are answer-until-correct, such that the student cannot proceed to the next question until he answers the current one correctly. Students have unlimited attempts per question. At a weekly *lab*, students receive zero or more question sets and receive credit if they complete the question set. This assessment setting results in many responses. Every response submitted by a student is recorded and timestamped.

We used the TAGS data to build a student error model that determined when to deliver hints to students. To summarize that work, first here are important terms.

- *Machine-marked-wrong answer (MMWA)* - A (question, string) pair OK marks as incorrect.
- *Response* - A (student, MMWA) pair.
- *Wrong answer* - A MMWA that is confirmed incorrect, as opposed to a false positive marked incorrect by OK.
- *Tag* - Human experts' interpretation of a specific student difficulty that could lead to an observed student error.
- *(Un)Inspected MMWA* - Whether a MMWA has gone through the tagging process in Stephens-Martinez et al. [18].

The TAGS data includes the entire corpus of inspected and uninspected MMWAs. The inspected MMWAs are divided into two sets. The *FrequentSet*, which consists of each question set's most frequent MMWAs that cover a total of 60% of the question set's responses and all MMWAs that individually cover 0.4% of the responses. The *StudentSet* that includes all MMWAs from a random 50 student subsample per question set. Each MMWA in these sets is categorized as either: (1) *conceptually correct* - it is marked wrong due to a typo, (2) *not an answer* - it is marked wrong but was not intended as an answer, or (3) *student error* - it is marked wrong due to conceptual errors or carelessness. In addition, if a MMWA is categorized as student error, it is assigned zero or more tags.

4 STUDENT ERROR MODEL

We created a model that inferred information about the student using the TAGS data's tagged MMWAs and quantitative metrics about the entire corpus. We infer information because, while tagging the frequent MMWAs covers a majority of the data set, it is only ≈5% of all MMWAs [18]. Tagging solely frequent MMWAs means most MMWAs are not inspected and therefore not tagged. Also, inspecting more is not necessarily worth the time and human resources. Therefore, all of a student's uninspected MMWAs are potentially lost information about that student.

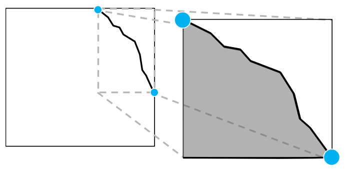
Figure 1: Graphical explanation modified AUC.

We define a student as having a tag if she has two wrong answers that share a tag. We chose two wrong answers because it allowed for students to slip and not receive a hint prematurely. How many wrong answers to ideally require we leave to future work. We use co-occurrence, inspired from pattern mining, to infer information by propagating tags. Co-occurrence measures in a single statistic how often two things happen together, happen alone, or neither happens. Each co-occurrence metric emphasizes different aspects of these occurrences. The metrics we considered include coherence, cosine, and Kulczynski explored in Wu et al.'s [23] pattern mining work, as well as the odds ratio. Using this definition and co-occurrence our model follows two rules to tag students:

- *Primary rule*: Two wrong answers that share the tag OR
- *Propagation rule*: There is only one wrong answer for the tag and an uninspected MMWA whose co-occurrence metric with the tagged wrong answer is above a threshold.

The model applies tags to students at the lab level. We calculate at the lab level (1) because a lab's question sets are usually related and (2) to use all data within a student's code-tracing session.

To understand how well co-occurrence could help our model, we compared it to an uninformed baseline. The uninformed baseline replaced the co-occurrence threshold with a probability to apply the tag to the student. We compared the models by plotting their precision vs. recall curve and calculating a modified area under the curve (AUC). For each model, we used the FrequentSet as the training set and the StudentSet as the test set. To calculate the precision and recall we defined a positive tagging as the model properly applying a tag to a student in the StudentSet and a negative as properly not applying a tag to a student in the StudentSet. We give a positive and negative assignment only within the context of the tags known to appear in the question set(s) under consideration.

Figure 1 shows a graphical explanation of why and how we modified the AUC. We made this modification because the primary rule and our definition of how a student receives a tag caused high precision and recall values, represented by the left box in Figure 1. These high values made it unclear the meaning of a large portion of the figure's area. For example, one question set had precision and recall values between [0.64, 1.0] and [0.71, 0.90] respectively. Therefore, as seen in Figure 1, we defined our modified AUC by first anchoring each curve at its extreme points (blue circles) and then scaling the curve such that it fits inside a 1x1 square. This definition allows us to compare the models within a question set because each

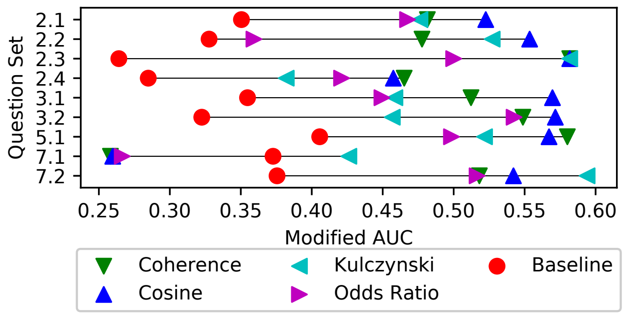

Figure 2: Dot plot comparing the modified AUC for each question set across the metrics and baseline. Note: The x-axis range is not 0 to 1 and the modified AUC cannot be compared between question sets.

model's endpoints are the same due to the propagation rule being completely permissive or intolerant. However, the modified AUC cannot be compared between question sets.

Figure 2 shows a dot plot comparing the modified AUCs for each question set. Each question set's decimal number represents first the lab it was in (left of the decimal) and its order in the lab (right of the decimal). From this figure, we can see all models for all question sets except Question Set 7.1 outperform the uninformed baseline. This outperformance leads us to conclude that co-occurrence yields useful information on what tags to apply to a student. However, when comparing the ranking of the metrics for each question set, we find no consistent ordering. This inconsistency leads us to conclude the best metric is question set specific.

Therefore, to create a question set's model we did the following:

(1) Choose the metric with the best modified AUC
(2) Calculate the f2-score for all that metric's threshold values
(3) Use the threshold with the best f2-score

5 KINDS OF HINTS

Our reteaching hints (N=75) re-explain a concept (*e.g.*, if...else) or an idea (*e.g.*, error) to the student. To give students reteaching hints, we associate a tag with one or more concepts. We write a reteaching hint for each concept, which means a tag may have multiple hints.

We based our KI hints (N=115) on the second component of Gerard et al.'s KI hints in science learning [6]. Our hints either (1) remind the student of an idea(s) or (2) ask the student to compare ideas. An idea, in this case, is a concept, segment of code, syntax, and anything else having to do with code-tracing. We wrote and assigned hints to (tag, wrong answer) pairs. However, most tags are associated with only one hint regardless of the wrong answer, and some tags share hints.

For example, we have a tag indicating students have "swapped the meaning of boolean and with or." This tag has three associated concepts: boolean and, or, and short-circuiting. When the model assigns this tag to a student, if he is receiving reteaching hints he would get three, including the first hint below. If he is receiving KI hints, he would receive only the second hint below.

```
|and| either returns the first false value evaluated
in an expression or the last value of that expression
if all the operands are true.
```

```
Remember the difference between |and| and |or|.
```

We placed |'s or bars around text meant to be code because the text is displayed in a terminal, where we could not use text formatting to highlight code. A full list of hints can be found in the appendices of Stephens-Martinez's work [17].

2.2: Loops	3.2: Higher Order Functions

`>>> def go(n):`
`...     if n % 2 != 0:`
`...         print(n / 2)`
`...         return`
`...     while n > 0:`
`...         print(n)`
`...         n = n // 2`
`>>> go(4)`
`4`
`2`
`1`
	`>>> def even(f):`
	`...     def odd(x):`
	`...         if x < 0:`
	`...             return f(-x)`
	`...         return f(x)`
	`...     return odd`
	`>>> stevphen = lambda x: x`
	`>>> stewart = even(stevphen)`
	`>>> stewart`
	`Function`
	`>>> stewart(61)`
	`61`

Figure 3: Examples of questions in the Question Sets.

6 EXPERIMENTAL SETTING

We ran our experiment during the 2017, 15-week, January to May academic term in University of CA, Berkeley's, large-enrollment introductory CS course (CS61A). The experiment took place in the weekly lab with self-paced computer exercises and teaching staff present. We administered hints in the code-tracing question sets delivered by OK. Labs usually covered material students have not yet used in practice. The teaching staff designed the code-tracing question sets to be the student's first interaction with the new material, and they are meant to be easy. An example of questions from Question Set 2.2, Loops, and 3.2, Higher Order Functions, are in Figure 3. We had 9 question sets spread over 4 labs. Question Sets 2.3 and 2.4 were not required.

6.1 Treatment Groups

We had three treatment groups: *control* with no hints, *reteaching*, and *KI*. Students were randomly assigned a treatment at the beginning of the course and kept the same treatment throughout the course. Due to a bug in our deployment, about three times as many students were in our control group (N=630) than in each hint group (reteaching N=215, KI N=212).

6.2 Question Sets and Models

To create the model that identifies when to give students hints, the model's parameters of co-occurrence metric and threshold were first determined per question set, see Table 1. These parameters indicated which wrong answer pairs had a high enough co-occurrence with each other. Then, when tagging students, the model took into account all wrong answers regardless of their original question set. Since we only considered the co-occurrence within a given question set, the propagation rule only worked within it.

Question sets had two subsets. First, in the *intervention questions* (N=77) – the original set of questions designed by the teaching staff

Table 1: Each question set's model parameters and statistics. The model used the metric represented by the letters under "Threshold": (Cos)ine, (Coh)erence, and (Kul)czynski. P and R stand for precision and recall

Question Set	Threshold		AUC	F2	P	R
2.1 Short Circuiting	Cos	0.082	0.523	0.930	0.746	0.990
2.2 Loops	Cos	0.130	0.554	0.879	0.823	0.895
2.3 Booleans	Cos	0.189	0.582	0.938	0.938	0.938
2.4 if...else	Coh	0.020	0.465	0.712	0.582	0.754
3.1 Lambdas	Cos	0.229	0.570	0.862	0.744	0.898
3.2 Higher Order Functions	Cos	0.204	0.572	0.815	0.745	0.835
5.1 List Comprehension	Coh	0.039	0.580	0.851	0.842	0.853
7.1 OOP	Kul	0.504	0.426	0.918	0.871	0.931
7.2 OOP	Kul	0.503	0.594	0.835	0.734	0.865

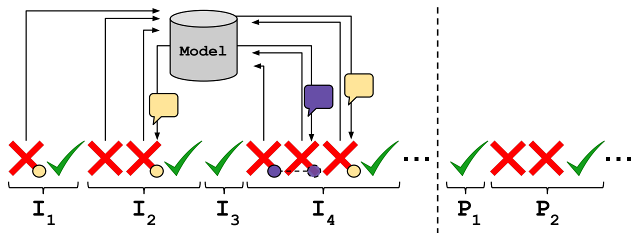

Figure 4: Hint delivery workflow.

– students could receive hints. Second, in the *post-test questions* (N=37), students would not receive hints. We placed these post-test questions after the intervention questions. Since the question sets are answer-until-correct, the students only answer them after receiving our hint intervention. We designed the post-test questions to surface the tags that appeared in the intervention questions.

6.3 Workflow: Delivering Hints to a Student

Figure 4 shows the workflow for delivering hints. As the student interacts with OK, he is submitting wrong answers (red X's) and correct answers (green checkmarks). The letters on the bottom show which question the answers were for and whether the question is an intervention (I) or post-test (P) question. OK gives the wrong answers to the model (grey cylinder), which updates its tagging of the student with each new wrong answer. If the latest wrong answer has a tag (circles) that the model has also given to the student, OK delivers to the student the hint(s) (speech bubbles) associated with that tag and wrong answer according to his treatment group. If the current wrong answer is uninspected, but it causes the propagation rule to apply a tag to the student, the student receives the hint(s) associated with the other inspected wrong answer (purple tags/hint). Students do not receive hints during the post-test questions nor does the model receive their wrong answers.

7 RESULTS

1,057 students attempted at least one question set and 689 students did all seven required question sets. When we split the students that did all required question sets by treatment group, control had 411 (65.2%), reteach had 145 (67.4%), and KI had 133 (62.7%) students. These numbers seem low because we lost students per question set due to missing information or computer error.

7.1 Measuring Performance

As far as we can determine, there is no consensus at present as to how to best measure student performance on answer-until-correct questions [2, 3, 22]. The following is the method we used.

First, measure the difficulty D_i of question i as 1 minus the fraction of students that correctly answered question i on the first attempt. Second, for a student, we have the binary value $C_i = \{0, 1\}$ of whether the student correctly answered question i on the first attempt. Finally, we define the performance of a specific student for a set of questions Q as:

$$\frac{\sum_{i \in Q} C_i \times D_i}{\sum_{i \in Q} D_i} \tag{1}$$

Our method was designed with three factors in mind. First, weighting by D_i rewards students for answering harder questions correctly. Second, using C_i emphasizes what the student knew initially, which has fewer confounds. Third, we divide by the weighted sum to normalize to only the questions the student attempted.

7.2 No Hints Versus Any Hint

Before investigating the difference between reteaching and KI hints, the first question we considered is whether receiving any hint was better than receiving no hint. To do this, we combined our reteaching and KI treatment groups into one "any hint" group. Then, we performed a Welch Two Sample t-test with the null hypothesis there is no difference between the performance on the post-test questions by students in the control group versus the students who received any hint. The test resulted in $p = 0.64$, meaning we cannot reject the null hypothesis. Since there is no statistically significant difference between these two groups, there was no need to investigate further distinctions between our two hint types.

8 EXPLORATORY ANALYSIS OF DATA

To understand why our results are null, we performed a retrospective, exploratory analysis. Our goals were to investigate: (1) if the system or students were performing any undesirable behaviors and (2) if there were hints that seemed to helped. To achieve these goals we, first, qualitatively analyzed students' wrong answer sequences. Second, we calculated the percent of times there were zero, one, or more MMWAs after a hint or MMWA from a set.

8.1 Undesirable Behaviors

We inspected a sample of long wrong answer sequences within a student and question set. We chose our sample by taking the (student, question set) tuples with the highest count of hints and wrong answers. We found three undesirable behaviors.

Human-Computer Interaction (HCI) problems in the hints and OK caused wrong answers. An HCI problem caused by our hints were

Table 2: Statistics when looking at the number of MMWAs after a (student, question, event) tuple, where the event is from a set of hint(s) or MMWAs.

	Count	% 0	% 1	% >1
MMWA Set				
Frequent Control	32,536	41.7	20.3	38.0
Frequent Reteaching	10,338	43.5	20.3	36.3
Frequent KI	11,324	41.5	20.3	38.1
Infrequent Control	17,651	32.6	19.8	47.6
Infrequent Reteaching	5,791	33.3	20.0	46.8
Infrequent KI	6,758	30.5	18.9	50.6
Hint Set				
Reteaching	4,072	49.1	18.1	32.7
KI	4,473	42.6	20.2	37.2
Correct/Typo	64	84.4	9.4	6.3
Reteaching range function	76	64.5	18.4	17.1
KI range function	68	57.4	23.5	19.1
KI short circuiting	44	56.8	27.2	15.9
KI for stacked calls cause errors	52	23.1	15.4	61.5
Reteaching Assignment	188	26.1	17.1	56.9

wrong answers that were correct except included bars, such as "`|False|`". A chief system HCI confusion was the use of answer keywords to indicate if an error occurred ("Error") or Python would display nothing ("Nothing") or a function pointer ("Function"). Another system problem is no text canonicalization.

Student brute-force guessing can create long MMWA sequences. A primary cause of long MMWA sequences was students systematically guessing. We found number sequences of 0 to 10, different orderings of `OK`'s answer keywords, and problem text snippets.

The model can propagate so aggressively that uninspected wrong answers received many tags and therefore many hints. For example, a student in the reteaching condition received six hints. A KI condition student received ten. The number of hints for a single answer in the KI condition especially surprised us because we never intended more than one KI hint to appear for a wrong answer.

8.2 Do Any Hints Help?

We attempted to find helpful hints by counting the number of MMWAs after a (student, question, event) tuple. The events were the occurrence of a hint or MMWA from a set. We then created summary statistics of these counts, see Table 2. The first column describes the contents of the MMWA or hint set. The "Count" column is the number of times a tuple occurred. The last three columns are the percent of those counts where after the event there were zero MMWAs (*i.e.,* the next answer was correct) "% 0", one MMWA "% 1", or more than one MMWA "% >1." Note, the way we are counting the tuples means that a frequent wrong answer (an event) could happen multiple times within a (student, question) pair.

To understand if any hints might have helped, we first calculated these statistics for the general case of all MMWAs in six different sets: frequent vs infrequent MMWAs crossed with the three treatment groups. These statistics are the two groups in the upper part of Table 2. The counts for the control group are three times as high because there were three times as many students in that group. After the general case, we investigated sets of hints. Below are some interesting potential findings for further investigation.

Students submit fewer MMWAs after a frequent MMWA compared to infrequent MMWAs. When comparing the "% 0" column, there is an average of 10.1% more for a frequent MMWA than an infrequent one across all three treatment groups. This increase seems almost entirely taken from the "% >1" column. One potential explanation for this increase is the brute-force guessing we saw in our case study. Most of the MMWAs from brute-force guessing are infrequent MMWAs. This majority means a student is submitting many infrequent MMWAs for a single question. Therefore, a given infrequent MMWA is likely to have many MMWAs after it.

Reteaching hints may help students submit fewer MMWAs than KI hints. The first group in the lower table is the statistics when a student received a reteaching or KI hint. By comparing these statistics to our frequent MMWA set's statistics, one can see that reteaching hints may help students submit fewer MMWAs because the "% 0" column increased, while KI hints are very similar. However, further work is needed to evaluate this possibility. We need to compare these statistics to the same set of statistics for the control group where a student would have gotten a hint but did not.

A hint suggesting the student has a typo for conceptually correct wrong answers considerably increased the "% 0" column. For both hint treatment groups, we delivered the hint "Typo?" for conceptually correct wrong answers. Comparing the statistics for this hint to the general reteaching and KI hint groups indicates that it drastically reduced the number of MMWAs. Once again, however, we do not know if this is causal without comparing it to the control group. Moreover, we only delivered this hint 64 times because most students did not submit enough wrong answers and the model was too strict by requiring two conceptual correct wrong answers.

Some hints decrease and others increase the "% 0" column. The next two groups of hint sets are the statistics for single hints. The first group is hints that decreased the number of MMWAs after the hint and the second group increased the number compared to the general statistics for each hint set. The difference is especially apparent when comparing the percentages in the "% >1" column to the overall reteaching and KI hint set. Our other hints spanned the values between these hints. It is currently not clear why specific hints performed better than others.

9 VALIDITY THREATS AND FUTURE WORK

While our overall results on improving student performance on post-test questions are null, we did find hints that decreased the number of attempts on the question. Potential threats to our validity that caused our null results include:

- We need a different performance metric, such as methods found in related work [2, 3, 21, 22].
- There is a topping out effect because the questions are too easy, which was the teaching staff's design goal for the original questions that we used as intervention questions.
- We did not have enough post-test questions to measure the hints' effect. The teaching staff limited the question sets' lengths. Therefore, to have sufficient questions during the

intervention portion, we had few post-test questions and designed the questions to test for multiple tags at once.
- Our KI hints and the learning setting and domain differ from related work that used KI guidance [6, 19, 21].
- Regarding our reteaching hints, there are also experimental setting differences between our and related work [4, 10].

For future work, we plan (1) a further investigation to understand our current results and (2) to improve the system or experiment.

To understand our results we will conduct user studies to investigate if the hints are understandable. Moreover, there are still more metrics we can mine from the existing data, such as the time between answers to gauge if students actually read the hints, like in Heckler and Mikula [8] and other performance metrics, like Ahadi et al. [1] using the $\log_2$ of the number of attempts by a student. Another area for future work would be to tag the wrong answers in the post-test questions and then compare the tags that the model assigned to the student with the student's post-test question tags.

To improve the system we will disincentivize brute-force guessing, address the HCI problems, and improve questions. We will improve the experiment by comparing the current model with a stricter model that does not propagate, immediately deliver the "Typo?" hint, limit the number of hints per response, measure prior knowledge, and investigate more kinds of formative feedback.

10 CONCLUSION

We set out to investigate ways to deliver hints to students based on information about their wrong answers, as well as comparing two different kinds of hints. To do this, we conducted an in situ experiment that delivered hints to 1,057 students based on their machine-marked-wrong answers (MMWAs) as they answered 9 question sets during a 15-week course. We found that co-occurrence yields useful information about MMWAs without individually inspecting them. We then used this information to create a student error model to determine when to deliver hints to students. However, we were not able to find a difference in student performance on post-test questions between our treatment groups.

Our retrospective, exploratory analysis to understand our null results led us to find: (1) HCI problems, (2) that students create long MMWA sequences due to brute-force guessing, and (3) our model is sometimes too aggressive in delivering hints. In addition, we found preliminary evidence that: (1) a student is more likely to get a question correct after submitting a frequent MMWA than an infrequent one, (2) students correctly answer a question in fewer MMWAs after a reteaching hint than a KI hint, (3) the simplistic hint "Typo?" for conceptually correct wrong answers potentially drastically reduces the number of MMWAs after it, and (4) both reteaching and KI vary in how many MMWAs the student submits between receiving the hint and submitting the correct answer.

ACKNOWLEDGMENTS

We would like to acknowledge the undergraduate researchers that supported this work but not sufficiently to be authors. Krishna Parashar and Regina Ongowarsito spent a summer writing hints and contributing to the building of the model. Kavi Gupta and Nikunj Jain monitored and supported the running of the experiment itself. Moreover, Nikunj contributed to the preliminary data processing.

REFERENCES

[1] Alireza Ahadi, Raymond Lister, and Arto Vihavainen. 2016. On the Number of Attempts Students Made on Some Online Programming Exercises During Semester and Their Subsequent Performance on Final Exam Questions. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, New York, NY, USA, 218–223.

[2] Yigal Attali. 2011. Immediate Feedback and Opportunity to Revise Answers. *Applied Psychological Measurement* 35, 6 (2011), 472–479.

[3] Yigal Attali and Don Powers. 2010. Immediate Feedback and Opportunity to Revise Answers to Open-Ended Questions. *Educational and Psychological Measurement* 70, 1 (2010), 22–35.

[4] Paul Baffes and Raymond Mooney. 1996. Refinement-based student modeling and automated bug library construction. *Journal of Interactive Learning Research* 7, 1 (1996), 75.

[5] Soumya Basu, Albert Wu, Brian Hou, and John DeNero. 2015. Problems Before Solutions: Automated Problem Clarification at Scale. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale (L@S '15)*. ACM, New York, NY, USA, 205–213.

[6] Libby F Gerard, Kihyun Ryoo, Kevin W McElhaney, Ou Lydia Liu, Anna N Rafferty, and Marcia C Linn. 2016. Automated guidance for student inquiry. *Journal of Educational Psychology* 108, 1 (2016), 60.

[7] John Hattie and Helen Timperley. 2007. The Power of Feedback. *Review of Educational Research* 77, 1 (2007), 81–112.

[8] Andrew F Heckler and Brendon D Mikula. 2016. Factors affecting learning of vector math from computer-based practice: Feedback complexity and prior knowledge. *Physical Review Physics Education Research* 12, 1 (2016), 010134.

[9] Peter Holt, Shelli Dubs, Marlene Jones, and Jim Greer. 1994. *The State of Student Modelling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 3–35.

[10] Amruth N Kumar. 2006. Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. *Technology, Instruction, Cognition and Learning (TICL) Journal* 4, 1 (2006).

[11] Nguyen-Thinh Le, Sven Strickroth, Sebastian Gross, and Niels Pinkwart. 2013. A review of AI-supported tutoring approaches for learning programming. In *Advanced Computational Methods for Knowledge Engineering*. Springer, 267–279.

[12] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further Evidence of a Relationship Between Explaining, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '09)*. ACM, New York, NY, USA, 161–165.

[13] 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)*. ACM, New York, NY, USA, 101–112.

[14] B Jean Mason and Roger H Bruning. 2001. Providing feedback in computer-based instruction: What the research tells us. In *CLASS Research Report*, Vol. 9. Center for Instructional Innovation.

[15] Edna H Mory. 2004. Feedback research revisited. *Handbook of research on educational communications and technology* 2 (2004), 745–783.

[16] Valerie J. Shute. 2008. Focus on Formative Feedback. *Review of Educational Research* 78, 1 (2008), 153–189.

[17] Kristin Stephens-Martinez. 2017. *Serving CS Formative Feedback on Assessments Using Simple and Practical Teacher-Bootstrapped Error Models*. Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-166.html

[18] Kristin Stephens-Martinez, An Ju, Krishna Parashar, Regina Ongowarsito, Nikunj Jain, Sreesha Venkat, and Armando Fox. 2017. Taking Advantage of Scale by Analyzing Frequent Constructed-Response, Code Tracing Wrong Answers. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 56–64.

[19] Charissa Tansomboon, Libby F. Gerard, Jonathan M. Vitale, and Marcia C. Linn. 2017. Designing Automated Guidance to Promote Productive Revision of Science Explanations. *International Journal of Artificial Intelligence in Education* 27, 4 (01 Dec 2017), 729–757.

[20] Anne Venables, Grace Tan, and Raymond Lister. 2009. A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*. ACM, New York, NY, USA, 117–128.

[21] Jonathan M. Vitale, Elizabeth McBride, and Marcia C. Linn. 2016. Distinguishing complex ideas about climate change: knowledge integration vs. specific guidance. *International Journal of Science Education* 38, 9 (2016), 1548–1569.

[22] Yutao Wang and Neil Heffernan. 2013. Extending knowledge tracing to allow partial credit: Using continuous versus binary nodes. In *International Conference on Artificial Intelligence in Education (AIED '13)*. Springer, 181–188.

[23] Tianyi Wu, Yuguo Chen, and Jiawei Han. 2010. Re-examination of interestingness measures in pattern mining: a unified framework. *Data Mining and Knowledge Discovery* 21, 3 (2010), 371–397.