An Evaluation of Data-Driven Programming Hints in a Classroom Setting

Thomas W. Price[1(✉)], Samiha Marwan[1], Michael Winters[1], and Joseph Jay Williams[2]

[1] North Carolina State University, Raleigh, USA
{twprice,samarwan,mawinter}@ncsu.edu
[2] University of Toronto, Toronto, Canada
williams@cs.toronto.edu

Abstract. Data-driven programming hints are a scalable way to support students when they are stuck by automatically offering suggestions and identifying errors. However, few classroom studies have investigated data-driven hints' impact on students' performance and learning. In this work, we ran a controlled experiment with 241 students in an authentic classroom setting, comparing students who learned with and without hints. We found no evidence that hints improved student performance or learning overall, and we discuss possible reasons why.

Keywords: Data-driven hints · Computing education

1 Introduction and Background

A fundamental challenge in computer science (CS) education is supporting novice students' learning as they work on independent programming practice. This practice is a common feature of CS courses, but it is challenging for novices working without instructor assistance [3,6,8]. To address this, researchers have designed adaptive, data-driven hints that help students right at the moment they are stuck by offering a personalized suggestion for how to progress or fix an error [14,19]. These are called *data-driven* hints because they are generated from prior students' data [5,11,12,19], allowing them to support diverse solutions [13,19] and scale to support any number of students with little additional instructor effort.

However, because they are generated from data, these hints only suggest how to progress, without the expert-authored explanations and domain principles found in many tutoring systems [22]. This suggests a need for careful evaluation of *data-driven* hints' impact on student performance and learning, especially in authentic classroom settings. However, most prior evaluations have used experts to evaluate the quality of these hints [7,11,12,15,17,23], rather than measuring their effect on learners. Studies that do so provide interesting yet inconclusive

I. I. Bittencourt et al. (Eds.): AIED 2020, LNAI 12164, pp. 246–251, 2020.
https://doi.org/10.1007/978-3-030-52240-7_45

results. One evaluation by Rivers suggests that data-driven programming hints in the ITAP tutoring system had little impact on student learning [18]. However, other work by Marwan et al. suggests that hints can promote learning, but only when carefully designed to scaffold self-explanation [10]. These mixed results not only suggest the need for further evaluation, but also that the effectiveness of hints may depend on their design and the learning context.

In this work, we investigated the efficacy of data-driven programming hints through a controlled study in an introductory CS course. We found that hints had no impact on overall learning or performance, which may have been due to specific choices in the design of hints and low hint quality for students with more complex mistakes.

2 Data-Driven Python Hints

In this study, we used the SourceCheck data-driven hint generation algorithm [12]. SourceCheck takes as input a database of correct student (or expert) solutions to a given problem. When a student asks for a hint, the algorithm identifies a solution that closely matches the structure of the student's current code and suggests small edits to the student's code to bring it closer to that solution. SourceCheck was originally developed for the block-based iSnap tutoring system [14], but in this work we have adapted it to generate hints for the Python programming language and integrated it into a new learning environment. We used students' solutions from a prior semester to generate hints for this study. In a prior technical evaluation, SourceCheck hints were found to be of high-quality compared to other data-driven hint generation approaches, on both iSnap and Python datasets [15], but they have not been evaluated in a large-scale classroom setting.

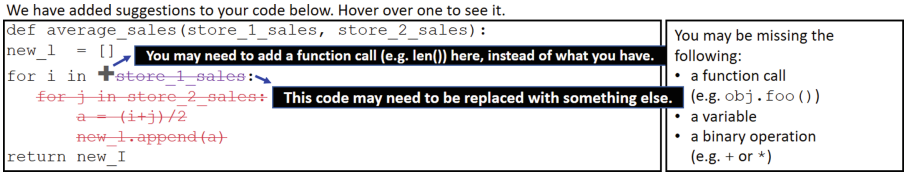

Fig. 1. An example of data-driven Python hints, annotating a student's code. (Color figure online)

In this study, hints were displayed by showing a copy of the student's code, annotated with three types of feedback, as shown in Fig. 1. The hints highlight code that should potentially be: 1) deleted (red strikethrough), 2) replaced (purple strikethrough), or 3) inserted (gray plus). Students can hover over these annotations for further explanation. Students are also shown a list of missing code elements. These suggestions do not directly give away solution code

(e.g. add "a boolean value" rather than "False"), reducing the possibility for bottom-out help-abuse [1,4,20]. We chose to provide multiple hints simultaneously, as prior work suggests that students often search through many hints to find one that addresses their current goals [16]. We also chose to show hints each time a student submitted their code to run and test it, given prior work suggesting that many students avoid asking for hints even when they need them [2,9,20].

3 Method

We investigated the following research question: What is the impact of data-driven programming hints on students' overall performance and learning?

Population: Our study took place in an in-person introductory Computer Science course at a large public university in North America, consisting of CS-majors and non-majors with little to no prior programming experience. Our study focused on an optional, online review assignment, which students were given in preparation for their final exam. The class included 1055 students, of whom 401 consented to their data being collected for research and 241 (60.1% of these) participated in the review assignment. Students were randomly assigned to either the Hint condition ($n = 119$), which received hints on some problems, or the Control condition ($n = 122$), which did not.

Procedure: During the review assignment, students completed 4 code writing tasks in an online practice environment. In each problem, students completed a function stub based on a brief description and examples of correct input/output. Each time students submitted their code, it was checked with 4–7 test cases, and the results were reported to the student. Students could submit as many attempts to a given problem as they wanted, revising their solution until it passed all test cases. The review assignment included 2 pairs of related problems (4–5 lines of code), respectively covering: 1) parallel list operations and 2) searching parallel lists. Each pair consisted of an "A" problem, where students could receive hints (depending on their condition), and a subsequent "B" problem, which was used for assessment and provided no hints for either condition. The B problem was a slightly more challenging version of the A problem and therefore allowed us to measure what students learned during the A problem. Students completed two A problems (1A, 2A), with hints in the Hint condition, and then the two corresponding B problems without hints (1B, 2B). The problems were of typical difficulty for the course, and they ranged from 32–46% of students getting them correct on the first attempt (compared to 35% for the average problem in the course).

Measures: We measured students' performance (1A, 2A) and learning (1B, 2B) on a given problem as the number of attempts that they made on that problem until they got it correct (i.e. passed all test cases). Since students almost always got the problem correct *eventually* (98.5% of the time), the number of attempts captures how much the student struggled in that process. It also captures how

much feedback they needed, since each attempt received feedback from the data-driven hints (when provided) and test cases (both conditions).

3.1 Results and Discussion

To address our RQ, we compared students' performance on practice problems (1A and 2A), where the Hint condition had hints, and on assessment problems (1B and 2B), which measured hints' impact on learning. As shown in Table 1, the averages are very similar for both conditions, and a Mann-Whitney U-tests show that the difference is not significant on any of the problems, with a small effect size. We also looked at the rate at which students correctly completed problems in each group, since prior work suggests data-driven hints can increase homework completion rates [21]. However, we found little difference between the overall completion rate of the Hint (90.8%) and Control (87.7%) conditions. This suggests that our data-driven hints did not have an overall effect on students' performance or learning.

Table 1. For each problem, and each condition, the mean number of attempts per student (lower is better), p-value from Mann-Whitney U test, effect size, and the number of students who completed the problem correctly.

	Mean attempts (SD)				Completed correctly	
Problem	Hint	Control	p	Cohen's d	Hint (n = 119)	Control (n = 122)
1A	2.26 (1.83)	2.35 (2.72)	0.51	−0.04	117 (98.32%)	122 (100%)
2A	2.79 (2.65)	2.70 (3.79)	0.25	0.03	117 (98.32%)	117 (95.90%)
1B	2.32 (1.95)	2.25 (1.72)	0.77	0.04	109 (91.60%)	111 (90.98%)
2B	3.00 (3.78)	2.64 (2.14)	0.39	0.12	108 (90.8%)	107 (87.70%)

This result contrasts somewhat with prior work, as Marwan et al. found that hints improved students' *immediate performance* (on problems with hints) [10], and Rivers also found suggestive evidence hints increased students' speed on practice problems [18]. We note that the *way* we designed our data-driven hints may have been responsible for some of these differences. For example, our implementation of data-driven hints did not include hand-authored textual explanations (as in [10]). Our results that data-driven programming hints alone did not improve students' *learning* agree with those of both Rivers and Marwan et al. [10,18]. These results may stem from limitations in *data-driven* programming hints in particular, which can be inaccurate or difficult to interpret [16,17]. A manual investigation of the hints offered during our study suggests that the adaptive hints varied across students and were not of equal quality. As in prior work [15], they appeared most useful for students with small mistakes, and may have been confusing for students far from a correct solution.

4 Conclusion and Future Work

This work provides insight into the effectiveness of data-driven programming hints, with additional evidence that these hints alone may not always promote learning, or even performance. The latter result is surprising, given that hints give away part of the correct solution, and it contrasts with prior work [10, 18]. This may be explained by our preliminary finding that hint quality varied across situations, which suggests the need for future work investigating whether contextual factors, such as student prior knowledge and problem difficulty, may mediate hints' usefulness.

References

1. Aleven, V., Koedinger, K.R.: Investigations into Help seeking and Learning with a Cognitive Tutor. In: Papers of the AIED 2001 Workhop 'Help Provision And Help Seeking In Interactive Learning Environments', pp. 47–58 (2001)
2. Aleven, V., Stahl, E., Schworm, S., Fischer, F., Wallace, R.: Help seeking and help design in interactive learning environments. Rev. Educ. Res. **73**(3), 277–320 (2003)
3. Altadmri, A., Kölling, M., Brown, N.C.C.: The cost of syntax and how to avoid it: text versus frame-based editing. In: CELT: COMPSAC Symposium on Computing Education & Learning Technologies; Part of COMPSAC 2016: The 40th IEEE Computer Society International Conference on Computers, Software & Applications (2016)
4. Baker, R.S., Corbett, A.T., Koedinger, K.R.: Detecting student misuse of intelligent tutoring systems. In: Proceedings of the International Conference on Intelligent Tutoring Systems, pp. 531–540 (2004)
5. Choudhury, R.R., Yin, H., Fox, A.: Scale-driven automatic hint generation for coding style. In: Proceedings of the International Conference on Intelligent Tutoring Systems, pp. 122–132 (2016)
6. Collier, S., Downing, M.: A qualitative analysis of students' understanding of conditional control structures. In: Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 1293–1293 (2019)
7. Hartmann, B., Macdougall, D., Brandt, J., Klemmer, S.R.: What would other programmers do? suggesting solutions to error messages. In: Proceedings of the ACM Conference on Human Factors in Computing Systems, pp. 1019–1028 (2010). https://doi.org/10.1145/1753326.1753478
8. Ko, A., Myers, B., Aung, H.: Six learning barriers in end-user programming systems. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 199–206 (2004)
9. Marwan, S., Dombe, A., Price, T.: Unproductive Help-seeking in Programming: what it is and how to address it? In: To be published in the Proceedings of the 25th Annual Conference on Innovation and Technology in Computer Science Education (2020)
10. Marwan, S., Jay Williams, J., Price, T.: An evaluation of the impact of automated programming hints on performance and learning. In: Proceedings of the International Computing Education Research Conference (2019)
11. Piech, C., Sahami, M., Huang, J., Guibas, L.: Autonomously generating hints by inferring problem solving policies. In: Proceedings of the second (2015) ACM Conference on Learning@ Scale, pp. 195–204 (2015)

12. Price, T., Zhi, R., Barnes, T.: Evaluation of a data-driven feedback algorithm for open-ended programming. In: International Educational Data Mining Society (2017)
13. Price, T.W., Dong, Y., Barnes, T.: Generating data-driven hints for open-ended programming. In: Proceedings of the International Conference on Educational Data Mining (2016)
14. Price, T.W., Dong, Y., Lipovac, D.: iSnap: towards intelligent tutoring in novice programming environments. In: Proceedings of the ACM Technical Symposium on Computer Science Education, pp. 483–488 (2017)
15. Price, T.W., Dong, Y., Zhi, R., Paaßen, B., Lytle, N., Cateté, V., Barnes, T.: A comparison of the quality of data-driven programming hint generation algorithms. Int. J. Artif. Intell. Educ. **29**(3), 368–395 (2019). https://doi.org/10.1007/s40593-019-00177-z
16. Price, T.W., Liu, Z., Catete, V., Barnes, T.: Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In: Proceedings of the International Computing Education Research Conference (2017)
17. Price, T.W., Zhi, R., Barnes, T.: Hint generation under uncertainty: the effect of hint quality on help-seeking behavior. In: André, E., Baker, R., Hu, X., Rodrigo, M.M.T., du Boulay, B. (eds.) AIED 2017. LNCS (LNAI), vol. 10331, pp. 311–322. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61425-0_26
18. Rivers, K.: Automated Data-Driven Hint Generation for Learning Programming. Ph.D. thesis, Carnegie Mellon University (2017). http://krivers.net/files/thesis.pdf
19. Rivers, K., Koedinger, K.R.: Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. Int. J. Artif. Intell. Educ. **27**(1), 37–64 (2015). https://doi.org/10.1007/s40593-015-0070-z
20. Roll, I., Baker, R.S.D., Aleven, V., Koedinger, K.R.: On the benefits of seeking (and avoiding) help in online problem-solving environments. J. Learn. Sci. **23**(4), 537–560 (2014)
21. Stamper, J.C., Eagle, M., Barnes, T., Croy, M.: Experimental evaluation of a automatic hint generation for a logic tutor. Int. J. Artif. Intell. Educ. **22**, 3–17 (2013)
22. VanLehn, K.: The behavior of tutoring systems. Int. J. Artif. Intell. Educ. **16**(3), 227–265 (2006)
23. Watson, C., Li, F.W.B., Godwin, J.L.: BlueFix: using crowd-sourced feedback to support programming students in error diagnosis and repair. In: Proceedings of the International Conference on Web-based Learning, pp. 228–239 (2012)