A Comparison of Two Designs for Automated Programming Hints

Thomas W. Price[1], Joseph Jay Williams[2], Samiha Marwan[1]
[1]North Carolina State University; [2]University of Toronto
twprice@ncsu.edu, williams@cs.toronto.edu, samarwan@ncsu.edu

ABSTRACT: A growing body of work has explored how to automatically generate hints for novice programmers. However, little research has explored what trade-offs exist between different ways of providing hints, how users perceive them, and how they can be combined. In this work, we present preliminary data from a study comparing different types of hint support in a block-based programming environment, focusing on code hints and explanatory text hints. We conducted a study in which crowd workers completed two programming tasks with different combinations of hint support. We found that both code hints and text hints are rated as very helpful by users, and showing both together is rated as the most useful. Only code hints improved users' performance during programming on the current task.

Keywords: Automated hints, intelligent programming support, novice programmers

1 INTRODUCTION

There is clear evidence that novices find programming to be a very difficult task to learn (Bennedsen et al., 2007; Watson & Li 2014), and researchers have developed a variety of intelligent support tools to assist them. Programming hints are a popular form of support (e.g. Perelman et al., 2014; Lazar et al., 2017; Yi et al., 2017), since they can be generated automatically, often using student data (Price et al., 2017b; Rivers & Koedinger, 2017), allowing them to scale to new problems and contexts. These automated hints are typically presented as next-step "code hints," which suggest an edit that the student should make to their program to bring it closer to a correct solution, allowing them to proceed when stuck. While small-scale studies suggest that code hints have the potential to resolve student difficulties (Price et al., 2017a), students can find code hints difficult to interpret without explanations (Price et al., 2017c). Others have pointed out that these "bottom-out" hints, which give away part of the correct solution, may not lead to learning (Aleven & Koedinger, 2016; Paaßen et al., 2018). While other types of automated hints have been proposed (Suzuki et al., 2017), little work has explored how different hint types compare and interact in the domain of programming.

In this paper, we present preliminary results from a larger study exploring the design space of programming hints. We evaluated both "code hints" and explanatory "text hints" that explain a domain concept and connect it to the current problem. We report results from a randomized experiment in which Mechanical Turk workers completed two simple programming tasks with different combinations of code and text hints. We find that code hints are rated as most helpful by users, and they can also improve users' performance on the current task. We find that text hints are also regarded as helpful, though less so than code hints, and they offer a complementary benefit to code hints. However, they do not appear to contribute to users' programming performance.

2 METHODS

Design of Hint Support: In this work, we build on an existing system called iSnap (Price et al., 2017a), a block-based, novice programming environment that supports students with hints and feedback. iSnap's hints are generated by the data-driven SourceCheck algorithm (Price et al., 2017b), which uses a database of correct solutions for a given problem to generate hints automatically. Like other data-driven hint-generation systems (e.g. Rivers & Koedinger, 2017; Piech et al., 2015; Paaßen et al., 2018), iSnap's hints are *next-step, edit-based* hints, suggesting a specific edit to the student's code which will bring them closer to a correct solution. In this study, we augmented iSnap's "code hints" with explanatory "text hints" that say not only *what* to do but also *why*. Similar to principle-based hints (Dutke et al., 2008) or teaching hints (VanLehn et al., 2005), these text hints explain a relevant programming concept and then connect it to the problem objectives. For example, for an assignment to write a procedure for drawing a polygon, one text hint reads, "The `repeat` block allows you to run the same code a fixed number of times, like drawing each side of a polygon." To add text hints to iSnap for a given problem, we tagged each block in the correct solution to that problem with one or more relevant text hints. Anytime iSnap would show a hint to add that block, iSnap shows the corresponding text hint, either alongside of in place of the code hint, depending on a user's condition (explained below). In this experiment, we generated hints using a comprehensive set of expert-authored solutions, rather than student solutions, as these have been shown to produce higher-quality hints (Price et al., 2018). We tagged each expert-authored solution with text hints to ensure that we supported a variety of solutions.

Population: We recruited 233 total crowd workers through Amazon's Mechanical Turk platform, which has been suggested as an appropriate alternative to university participants (Behrend et al., 2011; Kittur et al., 2008), including CS education research (Lee & Ko, 2015). We studied crowd workers because this allowed us to recruit a larger number of participants, collect fine-grained survey data, and give participants different, uneven levels of support – all of which are difficult in a real classroom setting. We analyzed data from 209 participants (excluding 24 participants due to data collection errors). We only recruited participants who attested to having no programming experience (no courses or workshops), and we paid users $4-7 to complete the study (varying the amount to increase the speed of recruiting). We did not collect any demographic information from participants.

Procedure: Participants first read through a short tutorial on programming in iSnap for approximately 5 minutes, which covered the user interface of iSnap and explained all programming concepts needed for the later programming tasks (loops, input/output and drawing) using a combination of text and short example animations. Next, participants worked on a programming task (Task 1) for 15 minutes, in which they were to create a program to draw a polygon with any number of sides (chosen by the user). During this programming task, each user was randomly assigned to a condition that determined what type of help iSnap provided for the whole task. iSnap either provided no help, code hints only, text hints only, or code and text hints together, for 4 total conditions. Additionally, for participants who received hints, their condition dictated whether or not they received prompts to self-explain the hints (Chi et al., 1994), but in this preliminary work we analyze these two sub-conditions together. While participants programmed, every two minutes iSnap interrupted them to take the action dictated by their condition (e.g. showing a code hint), and then asked them for their thoughts on the action (post-help survey). While this timed approach differs from the typical, on-demand way that users request hints in existing systems (Rivers & Koedinger, 2017), previous work shows many users will avoid or abuse help when they can request it on-demand (Aleven et al., 2016; Price et al., 2017c,d). Our timed approach ensures that each user receives hint support frequently and regularly, enabling us to collect more extensive data about perceptions of hints. After this first task, users completed a second 15-minute programming task (Task 2), which was similar to the first task but more challenging. During this task, users received help every 2 minutes, as in Task 1, but the type of help was randomized and independent of their Task 1 condition, allowing us to use it as a form of post-test.

Measures: The work reported here is part of a larger analysis of the experiment, which involved a number of survey measures. However, here we only focus on the post-help survey and users' performance on the two programming tasks. We measured the latter by defining 4 objectives for Task 1 and for Task 2 (e.g. "draw a shape" or "correctly get and use input from the user"), such that each objective was independent, and completing all 4 indicated successful completion of the whole task. We developed an automatic grader to determine which objectives participants completed and verified approximately half of the grades manually. Since users received help in Task 1 based on their condition, it is used as a measure of programming *performance*. In Task 2, the help users received was randomized and independent of their original condition, and it is therefore used as a measure of *learning*, since any differences in performance among the conditions can be attributed to knowledge gained as a result of different programming support on Task 1.

3 RESULTS

We investigated two research questions about the impact of code and text hints on users' outcomes:

RQ1: *How does the type of hint support that users received impact their perception of iSnap's usefulness?* After Task 1, each user rated how useful iSnap's actions were overall (from 0 to 10). Figure 1 shows the distribution of these ratings, organized by what type of help iSnap provided every 2 minutes.

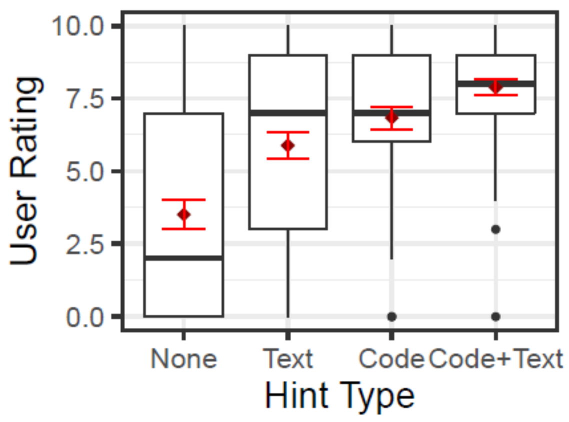

Figure 1: User ratings of how helpful they found iSnap's actions in Task 1, split by condition.

We analyzed users' usefulness ratings using a between-subjects ANOVA. We tested for the main effects of receiving code hints and text hints, as well as the interaction effect between code hints and text hints. We found a significant main effect for receiving code hints (*F*(1,205) = 30.3; *p* < 0.001) and text hints (*F*(1,205) = 17.1; *p* < 0.001), and we did *not* find a significant interaction effect between receiving code and text hints (*F*(1,205) = 2.57; *p* = 0.111). This suggests that users who received code or text hints in Task 1 rated iSnap's actions as significantly more useful than those who did not, and the lack of interaction suggests that there is an additive benefit to receiving *both* code and text hints. Confirming this, we found that the action usefulness ratings from users who received code hints *with* text hints (N=57; M=7.88; SD=2.08) were significantly higher than those who received *only* code hints (N=43; M=6.84; SD=2.53), as indicated by a Mann-Whitney *U*-test (*U*=903.5; *p* = 0.023).

RQ2: *How do code and text hints impact users' performance on current and future programming tasks?* Figure 2 shows the distribution of objectives completed by users on Tasks 1 and 2, split by the type of support they received *on Task 1*. For Task 1, we conducted an ANOVA on the number of objectives users accomplished, testing the main effects of receiving code hints and text hints and the interaction effect of receiving code hints and text hints. We found a significant main effect for receiving code hints (*F*(1,205) = 16.7; *p* < 0.001) but not text hints (*F*(1,205) = 0.052; *p* = 0.821), and we did not find a significant interaction effect between receiving code and text hints (*F*(1,205) = 0.003; *p* = 0.960). This suggests that only code hints contributed to users' programming performance on Task 1. The number of objectives completed was greater for users who received code hints (N=100; M=2.02; SD=1.50) than those who did not (N=109; M=1.20; SD=1.35), and the difference was significant (*U*=7118.5; *p*<0.001) with a medium effect size (Cohen's *d* = 0.575).

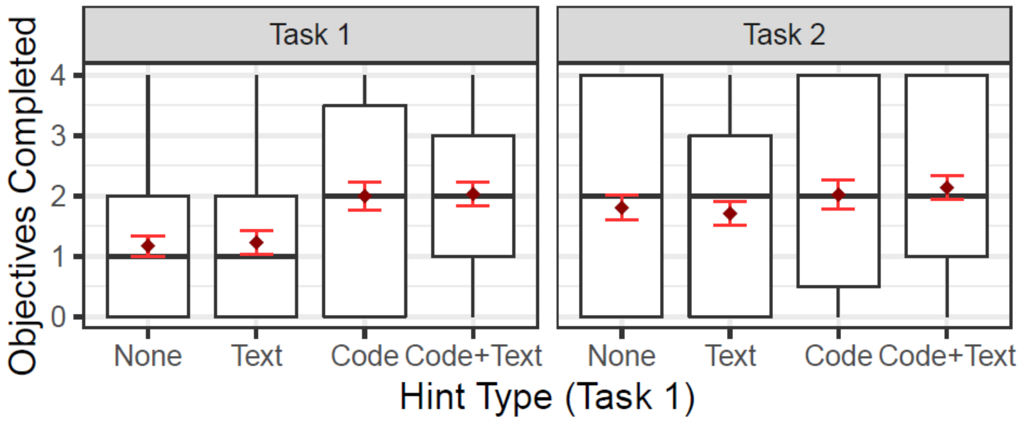

Figure2 : The number of objectives completed by users in Task 1 and Task 2, split by condition on *Task 1*. Box plots reflect medians and quartiles, and means (diamonds) with standard error bars.

We also investigated users' performance on Task 2 as a function of their original support condition *on Task 1*. We conducted an ANOVA to compare the main effects of having received code hints, text hints and reflective prompts on Task 1. We found no significant main effects for code hints (*F*(1,205) = 2.33; *p* = 0.128), text hints (*F*(1,205) = 0.003; *p* = 0.959), and no interaction between receiving code and text hints (*F*(1,205) = 0.253; *p* = 0.615). While not significant, there was still a small effect of code hints on the number of Task 2 objectives completed, as shown in Figure 2. The number of Task 2 objectives completed by users who received code hints in Task 1 (N=100; M=2.09; SD=1.51) was greater than those who did not (N=109; M=1.76; SD=1.51), but this difference was not significant (*W*=6115.5; *p* = 0.119) and had a small effect size (Cohen's *d* = 0.218).

4 DISCUSSION AND CONCLUSION

Our results show that learners find both code and text hints to be useful on their own, but the combination of both is perceived as the most useful. This suggests that both types of hints offer different information to the user, and both contribute the perceived utility of the hint. Code hints were also perceived as more useful than text hints, perhaps because they provided an immediately actionable suggestion, which all novices could follow, while text hints required users to interpret and apply the provided domain knowledge. In future work, we will investigate users' survey responses to better understand what benefits they perceived from each type of hint. Despite the perceived utility of both hint types, only code hints improved users' immediate performance, and they had no significant impact on future performance. Text hints did not improve users' performance, either alone or in conjunction with code hints. It is interesting that the perceived utility of text hints did not translate into improved student performance, and we hope to investigate other ways that these hints may have affected user outcomes in future work. It is possible that the impact of text hints would be more apparent over longer assignments. Further, if hints are provided on-demand, rather than every 2 minutes, hints that are perceived as more useful may result in more hint requests.

There are several limitations to this work. Our population consisted of paid crowd workers with no prior programming experience. Their motivations, prior knowledge and priorities may differ from those of other populations of learners where programming hints are used. Additionally, we only studied users during two simple, 15-minute programming tasks, and we have begun further work to investigate if our results generalize to longer or more complex tasks in classrooms. The presence of additional, randomized hints on Task 2 likely added noise to our performance data, making it more difficult to detect the effect of the Task 1 condition. The frequent delivery of hints allowed us to collect rich data about user's perceptions (which we will analyze in future work), but this proactive hint delivery differs from the usual on-demand approach, in which students request hints when needed.

REFERENCES

Aleven, V., Roll, I., McLaren, B. M., & Koedinger, K. R. (2016). Help Helps, But Only So Much: Research on Help Seeking with Intelligent Tutoring Systems. International Journal of Artificial Intelligence in Education, 26(1), 1–19. https://doi.org/10.1007/s40593-015-0089-1.

Behrend, T. S., Sharek, D. J., Meade, A. W., & Wiebe, E. N. (2011). The viability of crowdsourcing for survey research. Behavior Research Methods, 43(3), 800–813. https://doi.org/10.3758/s13428-011-0081-0.

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. ACM SIGCSE Bulletin, 39(2), 32. https://doi.org/10.1145/1272848.1272879.

Chi, M. T. H., De Leeuw, N., Chiu, M.-H., & LaVancher, C. (1994). Eliciting self-explanations improves understanding. Cognitive Science, 18(3), 439–477.

Dutke, S., & Reimer, T. (2008). Evaluation of two types of online help for application software. Journal of Computer Assisted Learning, 16(October 2000), 307–315. https://doi.org/10.1046/j.1365-2729.2000.00143.x.

Kittur, A., Chi, E. H., & Suh, B. (2008). Crowdsourcing user studies with Mechanical Turk. In Proceedings of the SIGCHI conference on human factors in computing systems (pp. 453–456).

Lazar, T., Možina, M., & Bratko, I. (2017). Automatic Extraction of AST Patterns for Debugging Student Programs. In Proceedings of the International Conference on Artificial Intelligence in Education (pp. 162–174). https://doi.org/10.1007/978-3-319-61425-0_14.

Lee, M. J., & Ko, A. J. (2015). Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. In Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15 (pp. 237–246). New York, New York, USA: ACM Press. https://doi.org/10.1145/2787622.2787709.

Paaßen, B., Hammer, B., Price, T. W., Barnes, T., Gross, S., & Pinkwart, N. (2018). The Continuous Hint Factory -Providing Hints in Vast and Sparsely Populated Edit Distance Spaces. Journal of Educational Data Mining, 1–50.

Perelman, D., Gulwani, S., & Grossman, D. (2014). Test-Driven Synthesis for Automated Feedback for Introductory Computer Science Assignments. In Proceedings of the Workshop on Data Mining for Educational Assessment and Feedback.

Piech, C., Sahami, M., Huang, J., & Guibas, L. (2015). Autonomously Generating Hints by Inferring Problem Solving Policies. In Proceedings of the ACM Conference on Learning @ Scale (pp. 1–10).

Price, T. W., Dong, Y., & Lipovac, D. (2017a). iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In Proceedings of the ACM Technical Symposium on Computer Science Education.

Price, T. W., Liu, Z., Catete, V., & Barnes, T. (2017c). Factors Influencing Students' Help-Seeking Behavior while Programming with Human and Computer Tutors. In Proceedings of the International Computing Education Research Conference.

Price, T. W., Zhi, R., & Barnes, T. (2017b). Evaluation of a Data-driven Feedback Algorithm for Open-ended Programming. In Proceedings of the International Conference on Educational Data Mining.

Price, T. W., Zhi, R., & Barnes, T. (2017d). Hint Generation Under Uncertainty: The Effect of Hint Quality on Help-Seeking Behavior. In Proceedings of the International Conference on Artificial Intelligence in Education.

Price, T. W., Zhi, R., Dong, Y., Lytle, N., & Barnes, T. (2018). The impact of data quantity and source on the quality of data-driven hints for programming. In Proceedings of the International Conference on Artificial Intelligence in Education. http://doi.org/10.1007/978-3-319-93843-1_35.

Rivers, K., & Koedinger, K. R. (2017). Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. International Journal of Artificial Intelligence in Education, 27(1), 37–64.

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

VanLehn, K., Lynch, C., Schulze, K., & Shapiro, J. A. (2005). The Andes physics tutoring system: Five years of evaluations. In Proceedings of the International Conference on Artificial Intelligence in Education.

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. In Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education (pp. 39–44).

Yi, J., Ahmed, U. Z., Karkare, A., Tan, S. H., & Roychoudhury, A. (2017). A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments. In Proceedings of the Joint Meeting on Foundations of Software Engineering (pp. 740–751). https://doi.org/10.1145/3106237.3106262.