The Hint Mechanism in Code Hunt

Daniel Perelman
University of Washington
perelman@cs.washington.edu

Judith Bishop
Microsoft Research
jbishop@microsoft.com

Sumit Gulwani
Microsoft Research
sumitg@microsoft.com

Dan Grossman
University of Washington
djg@cs.washington.edu

Abstract

How does one help students who become stuck on a problem? Specific hints for specific solutions can miss the point when one has thousands of students with different needs. We use data mining and program synthesis to generate hints that are tailored to the progress of the student. Our results indicate that we can provide accurate hints and also nudge the student along. Yet problems remain: should hints be given at all? Should students be able to turn them off? Do we have evidence that students read the hints? Is there a plateau after which the hint system will not improve even when more data is provided? Our work is based on Code Hunt from Microsoft Research, using data similar to the recently released public data set.

1. Introduction

Students sometimes get stuck on a problem, so feedback is a necessary part of the educational process. In order to teach students at scale we need to automatically produce feedback. But without experimentation and data we cannot know exactly what form this feedback should take.

In the Code Hunt programming game[5] we implemented a couple of new forms of feedback which we refer to as the hint system. Our "line hints" use program synthesis paired with data mining known solutions to find the nearest solution to a player's attempt and tell them they are close by letting them know which line they can change to reach a solution. Our "recommendation hints" use statistics on known attempts to determine if the player is using a method call associated with a dead-end solution strategy and warn them away from it or alternatively suggest a method call that will point them in the direction of the solution.

These two forms of feedback were designed based on our intuition of what might be helpful to a player and what we could meaningfully extract from the data we had.

But this raised multiple questions:

1. Is this is right form of feedback? Our intuition could be wrong. Maybe the general concept is right but our wording is poor? Maybe more or less detail would be better? Maybe some completely different form of feedback would be better?
2. When should this feedback be given? Can we guess when a player is struggling and give them a hint then? Or how long should we let the player struggle before they pass from feeling challenged to just feeling frustrated?
3. What should the UI look like? Currently hints are always shown as soon as they are generated. Maybe the player should have to request them? Or at least have an option to turn them off or otherwise not see hints they weren't expecting? Some players might be discouraged by seeing hints when they wanted to solve the puzzles without help.

We made a first attempt at answering these questions by running an A/B test where we disabled hints for some players and compared the behavior of players that saw hints versus those that did not. These initial results were positive—players that saw hints played the game for longer—but only give a small glimpse into answering those questions.

Luckily, with the platform we have we can run additional such tests on other aspects of our hint system once we have the right questions to ask.

2. Related work

There has been a recent burst of interest in automated feedback techniques for introductory computer science assignments.

Singh et.al. [3] take as input a reference solution and an error model, which consists of a set of rewrite rules inspired from common errors and corresponding fixes, and corrects an incorrect student submission using the SKETCH program synthesizer [4]. The feedback they generate is a set of changes to the program which they output in a few forms of varying detail (e.g. the exact change, just the location of the change, etc.).

Alur et. al. [1] restrict themselves to the easier problem of automated personalized feedback for finite automata. Unlike general programs, many properties of finite automata are computable, allowing them to generate fairly detailed feedback.

CodeWebs [2] takes a large number of submissions along with the results of a test suite on each one and groups them by syntactic similarity. By clustering submissions they can figure out which expressions are equivalent in the context of the assignment (even if they aren't equivalent in general) and other information that can be useful for generating assignment-specific feedback.

References

[1] R. Alur, L. D'Antoni, S. Gulwani, D. Kini, and M. Viswanathan. Automated grading of DFA constructions. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1976–1982. AAAI Press, 2013.

[2] A. Nguyen, C. Piech, J. Huang, and L. Guibas. Codewebs: Scalable homework search for massive open online programming courses. In *Proceedings of the 23rd International World Wide Web Conference (WWW 2014)*, 2014.

[3] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 15–26. ACM, 2013.

[4] A. Solar Lezama. *Program Synthesis By Sketching*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2008.

[5] N. Tillmann, J. Bishop, R. N. Horspool, D. Perelman, and T. Xie. Code hunt: Searching for secret code for fun. In *SBST*, 2014.