Data-driven Hint Generation from Peer Debugging Solutions

Zhongxiu Liu
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
zliu24@ncsu.edu

ABSTRACT

Data-driven methods have been a successful approach to generating hints for programming problems. However, the majority of previous studies are focused on procedural hints that aim at moving students to the next closest state to the solution. In this paper, I propose a data-driven method to generate remedy hints for BOTS, a game that teaches programming through a block-moving puzzle. Remedy hints aim to help students out of dead-end states, which are states in the problem from where no student has ever derived a solution. To address this, my proposed work includes designing debugging activities and generating remedy hints from students' solutions to debugging activities.

1. INTRODUCTION

Programming problems are characterized by huge and expanding solution spaces, which cannot be covered by manually designed hints. Previous studies have shown fruitful results in applying data-driven approaches to generate hints for programming problems. Barnes and Stamper [1] designed the Hint Factory, which gives student feedback using previous students' data. The Hint Factory uses a data structure called an interaction network as defined by Eagle et al. [3], in which nodes represent the program states and edges represent the transitions between states. Peddycord et al. [7] applied the Hint Factory in BOTS, a game that teaches programing through block-moving puzzles. This study introduced worldstates, which represent the output of a program, and compared them to codestates, snapshots of the source code. This study found that using interaction networks of worldstates can generate hints for 80% of programming states. Rivers and Koedinger [9] applied the Hint Factory in a solution space where snapshots of students' code (program state) are represented as trees, and trees are matched when the programs they represent are within a threshold of similarity. Piech et al. [8] applied data-driven approach to programs from a MOOC. This work compared the methods in Rivers and Koedinger's[9] and Barnes's[1] studies, together with algorithms that predict the desirable moving direction

from a program state and generate hints to push students toward the desirable direction.

However, previous studies mainly focused on generating procedural hints that direct students to the next program state. Data from previous students' work may be insufficient to provide a next-step hint from a "dead-end state". Second, even if a next-step hint could be generated, simply telling students where to move next is not enough. An example of this situation is shown in Figure 2 - if a student follows a path that leads to a dead-end state (marked in blue), then the only hint we are able to offer is to delete all work since the last branching point. This may be a bad advice; just because we have not seen a student solve the problem this way does not mean that the solution is incorrect. Even with a correct solution down this path, we are unlikely to see it since most students solved the problem in a more conventional way, either because they have a better understanding of the problem or because our hints guide them towards the more conventional solution. Thus, students in dead-end states, who may actually have a correct solution in mind, are unable to receive helpful hints.

In this paper, I propose a data-driven method to generate remedy hints in Bots. Remedy hints are hints that help students in dead-end states by telling them why their current state is wrong, and where to move from their current state. To address the problem of insufficient data, I will collect data from debugging activities in BOTS, where students work out solutions from dead-end states and provide explanations. I hypothesize that this study will not only help students who are wheel-spinning on dead-end states, but also the students who are providing debugging solutions.

2. RESEARCH METHODOLOGY
2.1 Designing Debugging Activities

Debugging activities will be designed as bonus challenges for students who successfully complete a level. The content of debugging activities will be the dead-end states from the problem they completed. Given a dead-end state, a student will first be asked to explain the error in the program, and why it led to the dead-end state. The student will then be asked to explain his/her debugging strategy. Lastly, the student will apply his/her debugging strategy and fix the program from its current state to a goal state. In this process, both the student-written explanations and the transitions of program states will be used as hints. A more detailed explanation of these are explained in the following section.
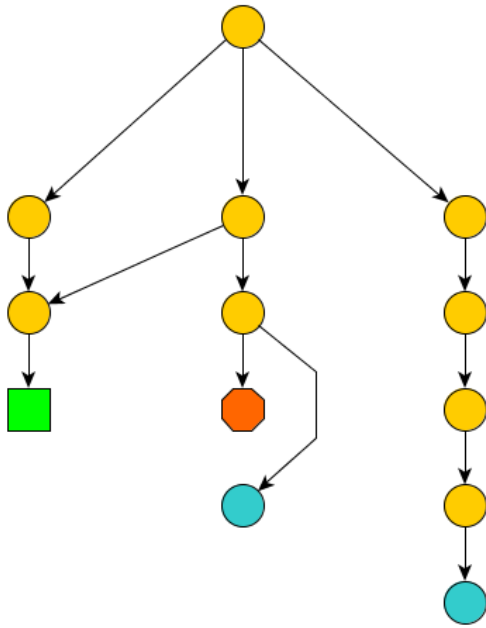
Figure 1: Interaction Network in BOTS.Green is the solution state;orange is an error state (e.g. the robot runs off the stage);blue are the dead-end states; yellow represents the rest states

To encourage students to participate in debugging activities, I will introduce a voting system. Completing debugging activities will earn points or advantages from the game. Currently, BOTS applies a rewarding system for students who solve the puzzle with fewer lines of code, as shown in Figure 2. On the left is the optimal number of lines of code needed to solve the puzzle. On the right is the current player's record for the fewest lines of code. Players earn 4 stars for reaching the optimal solution, 3 stars for being within a certain threshold value, down to one star for merely completing the puzzle. Additionally, clicking the optimal solution shows the name of the first user to reach the optimal solution.

I will design a similar leaderboard to reward students who used fewer steps when debugging for a dead-end state. Encouraging students to use fewer steps will reduce the size of debugging solutions, and the likelihood that a student will delete previous work and start from scratch. Moreover, students will receive rewards for writing good quality explanations on states and debugging strategies. The quality will be measured by a voting mechanism. Students who received a student-written explanation will be able to vote for the hint as "helpful." The more votes an explanation receives, the more points its author will get. Students with the most points will have their names appear in a leaderboard.

2.2 Construct Hint from Debugging Work
Completing a debugging problem is defined as successfully moving from the current state to the final goal state. The debugging process will be treated as a self-contained problem with its own local interaction network. When completed, this local interaction network will be added to the global interaction network for the problem. With a more complete
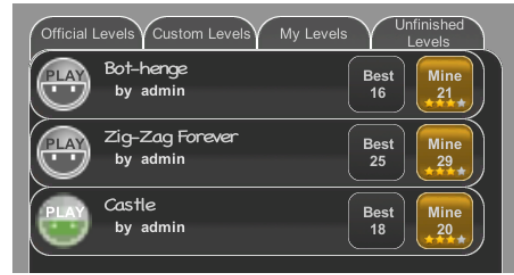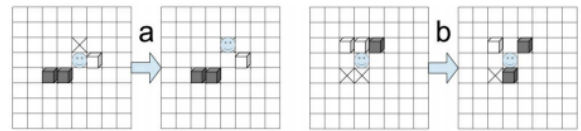

Figure 2: BOTS rewarding system

Figure 3: Two generated hints for a simple puzzle. The blue is the robot. The 'X' is a goal. Shaded boxes are boxes placed on goal spot. Not shaded boxes are not on goal spot.

global interaction network, Hint Factory [1] can be applied to generate hints for previously dead-end states.

Student-written explanations will be presented together with hints generated by the Hint Factory. An example hint from the current BOTS system is shown in Fig 3. Before presenting the hints from the Hint Factory, a student in dead-end state will see a student-written explanation on where and why their current program is wrong. This will give students a chance to reflect on their own program. Then, the student can request to see a student-written explanation of the debugging plan for the current state. This will enable the student to solve the problem on their own following a debugging plan, instead of blindly following procedural hints.

When multiple debugging approaches are available for a state, I will experiment with selecting the best debugging solution to generate hints. Ideally, I would select a debugging approach with the shortest solution path. However, there may be situations where students debugged by starting over from the beginning, which may or may not be the best solution. One approach is to evaluate the path that leads toward the current state. Assume there is a failure state in the student's solution; the earlier this failure state occurs in the path, the more likely the solution is wrong from the start and back-to-start is a good solution.

When multiple student-written explanations are available for a debugging solution, I will start by randomly choosing one explanation. As the voting process goes, I will filter out the explanations with significantly lower 'helpful' votes.

3. EVALUATION
My evaluation will focus on the below research questions:

- What percentage of students will participate in the debugging activities, and how many write explanations? Why do students participate or not?

- What is the relationship between students' involvement in debugging and their programming performance? Will students who complete problems with shorter solutions be more involved in debugging?

- Will writing or reading student-written explanations and debugging strategies help learning?

- In the global interaction network, what percentage of the dead-end program states receive hints from student debugging solutions?

Previous BOTS participants are students from after-school programming education activities. In my experiment, I will randomly recruit the same type of students. These students will be separated into a control group where students will use the traditional BOTS system, an experimental group A where students will be given the option to do debugging challenges, and an experimental group B where students must do debugging challenges after completing a level.

To answer the first research question, students from the two experimental groups will do a post survey on their opinions about debugging activities and hints generated from student-written explanations. For experimental group A, I will add survey questions on why students chose to participate or not participate in debugging activities. To answer the second question, students' interaction and compilation data while playing BOTS will be recorded. These data will be used to measure the relationship between involvement in the debugging activities and programming performance. To answer the third research question, students from all groups will do pre and post-tests on basic programming and debugging concepts that are related to BOTS content. Learning gains will be measured as the difference between pre and post-test. To answer the fourth question, the program state space coverage will be compared between the three groups.

4. PROPOSED CONTRIBUTION

My work will generate a new type of hint that may lead to different pedagogical results than the procedural hint, especially for students in dead-end states. My work will demonstrate the feasibility of collecting data from peer students' debugging processes, and generating helpful hints.

My work will design a feature that supports both programming and debugging activities in an educational game. This design will have several pedagogical benefits. First, Kinnunen and Simon's[6] research have shown that novice programmers experienced a range of negative emotions after errors. Practicing debugging will help novice programmers proceed after errors, and enjoy programming experiences. Second, students will make self-explanations on the observed flaw and debugging strategy, and decades of research such as Johnson and Mayer's[5], and Chi et al.[2] have shown that self-explanation is extremely beneficial to learning. Third, students in dead-end states will not only receive help, but also learn what peer students think given the same situation.

5. ADVICE SOUGHT

Johnson and Mayers[5], and Hsu et al. studies[4] have shown that merely adding self-explanation features did not help learning, but students' engagement in self-explaining did.

Therefore, I want to seek advice on the design of debugging activities that engage students in debugging and writing explanations, and produce quality work. I also want to seek advice on the evaluation. Given the previous question, how should I measure the level of engagement in debugging and self-explaining?

Moreover, introducing debugging challenges as extra activities will affect other measurements. For example, students who spend a significant amount of time in debugging may complete less problems given the time constraint, and exhaust earlier. How should I address this problem and measure students' performance fairly? Moreover, how to design pre and post-tests to measure learning gains from debugging process? Lastly, what are the potentials, benefits, and risks to expand this work into programming problems using mainstream programming languages?

6. REFERENCES

[1] T. Barnes and S. John. Toward automatic hint generation for logic proof tutoring using historical student data. In *Proceedings of the 6th International Conference on Intelligent Tutoring System*, pages 373–382, 2008.

[2] M. T. Chi, N. Leeuw, M. H. Chiu, and C. LaVancher. Eliciting self–explanations improves understanding. *Cognitive Science*, 18(3):439–477, 1994.

[3] M. Eagle, M. Johnson, and T. Barnes. Interaction networks: Generating high level hints based on network community clusterings. In *Proceedings of the 6th International Conference on Intelligent Tutoring System*, pages 164–167, 2012.

[4] C. Y. Hsu, C. C. Tsai, and H. Y. Wang. Facilitating third graders' acquisition of scientific concepts through digital game–based learning: The effects of self–explanation principles. *The Asia–Pacific Education Researcher*, 21(1):71–82, 2012.

[5] C. I. Johnson and R. E. Mayer. Applying the self-explanation principle to multimedia learning in a computer-based game–like environment. *Computers in Human Behavior*, 26(6):1246–1252, 2010.

[6] P. Kinnunen and B. Simon. Experiencing programming assignments in cs1: the emotional toll. In *Proceedings of the 6th international workshop on Computing education research*, pages 77–86, 2010.

[7] B. Peddycord III, A. Hicks, and T. Barnes. Generating hints for programming problems using intermediate output. In *Proceedings of the 7th International Conference on Educational Data Mining*, pages 92–98, 2014.

[8] C. Piech, M. Sahami, J. Huang, and L. Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of Learning at Scale*, 2015.

[9] K. Rivers and K. R. Koedinger. Automatic generation of programming feedback: A data-driven approach. In *Proceedings of the 1st workshop on AI-supported Education for Computer Science, th 16th International Conference on Artificial Intelligence on Education*, pages 50–59, 2013.