OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale

Elena L. Glassman
MIT CSAIL
elg@mit.edu

Jeremy Scott
MIT CSAIL
jks@mit.edu

Rishabh Singh
MIT CSAIL
rishabh@csail.mit.edu

Philip Guo
MIT CSAIL,
University of Rochester
pg@cs.rochester.edu

Robert C. Miller
MIT CSAIL
rcm@mit.edu

ABSTRACT

In MOOCs, a single programming exercise may produce thousands of solutions from learners. Understanding solution variation is important for providing appropriate feedback to students at scale. The wide variation among these solutions can be a source of pedagogically valuable examples, and can be used to refine the autograder for the exercise by exposing corner cases. We present OverCode, a system for visualizing and exploring thousands of programming solutions. OverCode uses both static and dynamic analysis to cluster similar solutions, and lets instructors further filter and cluster solutions based on different criteria. We evaluated OverCode against a non-clustering baseline in a within-subjects study with 24 teaching assistants, and found that the OverCode interface allows teachers to more quickly develop a high-level view of students' understanding and misconceptions, and to provide feedback that is relevant to more students.

Author Keywords

Data mining; programming exercises; MOOC

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: Graphical user interfaces

INTRODUCTION

Intelligent tutoring systems (ITSes), Massive Open Online Courses (MOOCs), and websites like Khan Academy and Codecademy are now used to teach programming courses at a massive scale. In these courses, a single programming exercise may produce thousands of solutions from learners, which presents both an opportunity and a challenge. For teachers, the wide variation among these solutions can be a source of pedagogically valuable examples [4], and understanding this variation is important for providing appropriate, tailored feedback to students [1, 3]. The variation can also be useful for refining evaluation rubrics, since it can expose corner cases in automatic grading tests.

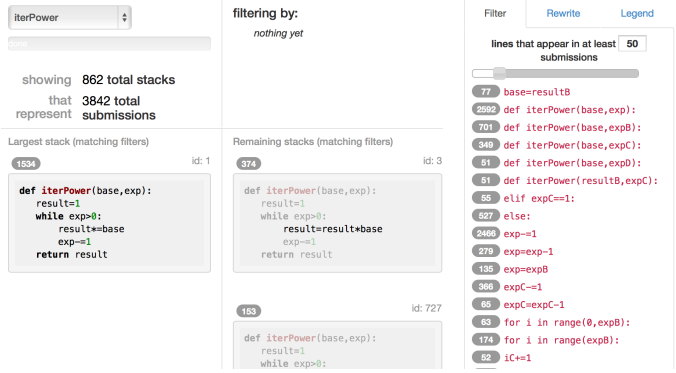

Figure 1. OverCode user interface.

Sifting through thousands of solutions to understand their variation and find pedagogically valuable examples is a daunting task, even if the programming exercises are simple and the solutions are only tens of lines of code long. Without tool support, a teacher may not read more than 50-100 of them before growing frustrated with the tedium of the task. Given this small sample size, teachers cannot be expected to develop a thorough understanding of the variety of strategies used to solve the problem, or produce instructive feedback that is relevant to a large proportion of learners, or find unexpected interesting solutions.

An information visualization approach would enable teachers to explore the variation in solutions at scale. Existing techniques [2, 3, 5] use a combination of clustering to group solutions that are semantically similar, and graph visualization to show the variation between these clusters. These clustering algorithms perform pairwise comparisons that are quadratic in both the number of solutions and in the size of each solution, which scales poorly to thousands of solutions. Graph visualization also struggles with how to label the graph node for a cluster, because it has been formed by a complex combination of code features. Without meaningful labels for clusters in the graph, the rich information of the learners' solutions is lost and the teacher's ability to understand variation is weakened.

UIST '14, Oct 05-08 2014, Honolulu, HI, USA.
ACM 978-1-4503-3068-8/14/10.
http://dx.doi.org/10.1145/2658779.2658809

In this poster we present OverCode, a system for visualizing and exploring the variation in thousands of programming solutions. OverCode is designed to visualize correct solutions, in the sense that they pass the automatic grading tests typically used in a programming class at scale. OverCode uses a novel clustering technique that creates clusters of identical cleaned code, is time linear in both the number of solutions and the size of each solution. The cleaned code is readable, executable, and describes every solution in that cluster. The cleaned code is shown in a visualization that puts code front-and-center (Figure 1). In OverCode, the teacher reads through code solutions that each represent an entire cluster of solutions that look and act the same. The differences between clusters are highlighted to help instructors discover and understand the variations among submitted solutions. Clusters can be filtered by the lines of code within them. Clusters can also be merged together with *rewrite rules* that collapse variations that the teacher decides are unimportant.

A cluster in OverCode is a set of solutions that perform the same computations, but may use different variable names or statement order. OverCode uses a lightweight dynamic analysis to generate clusters, which scales linearly with the number of solutions. It clusters solutions whose variables take the same sequence of values when executed on test inputs and whose set of constituent lines of code are syntactically the same. An important component of this analysis is to rename variables that behave the same across different solutions. The renaming of variables serves three main purposes. First, it lets teachers create a mental mapping between variable names and their behavior which is consistent across the entire set of solutions. This may reduce the cognitive load for a teacher to understand different solutions. Second, it helps clustering by reducing variation between similar solutions. Finally, it also helps make the remaining differences between different solutions more salient.

In two user studies with a total of 24 participants, we compared the OverCode interface with a baseline interface that showed original unclustered solutions. When using OverCode, participants felt that they were able to develop a better high-level view of the students' understandings and misconceptions. While participants did not necessarily read more lines of code in the OverCode interface than in the baseline, the code they did read came from clusters containing a greater percentage of all the submitted solutions. Participants also drafted mock class forum posts about common good and bad solutions that were relevant to more solutions (and the students who wrote them) when using OverCode as compared to the baseline.

The main contributions of this work are:

- a novel visualization that shows similarity and variation among thousands of solutions, with cleaned code shown for each variant.
- an algorithm that uses the behavior of variables to help cluster solutions and generate the cleaned code for each cluster of solutions.
- two user studies that show this visualization is useful for giving instructors a bird's-eye view of thousands of students' solutions.

SUMMARY

We have designed the OverCode system for visualizing thousands of Python programming solutions to help instructors explore the variations among them. Unlike previous approaches, OverCode uses a lightweight static and dynamic analysis to generate stacks of similar solutions and uses variable renaming to present cleaned solutions for each stack in an interactive user interface. It allows instructors to filter stacks by line occurrence and to further merge different stacks by composing rewrite rules. Based on two user studies with 24 current and potential teaching assistants, we found OverCode allowed instructors to more quickly develop a high-level view of students' understanding and misconceptions, and provide feedback that is relevant to more students. We believe an information visualization approach is necessary for instructors to explore the variations among solutions at the scale of MOOCs, and OverCode is an important step towards that goal.

ACKNOWLEDGMENTS

This material is based, in part, upon work supported by the National Science Foundation Graduate Research Fellowship (grant 1122374), the Microsoft Research Fellowship, the Bose Foundation Fellowship, and by Quanta Computer as part of the Qmulus Project. Any opinions, findings, conclusions, or recommendations in this paper are the authors', and do not necessarily reflect the views of the sponsors.

REFERENCES

1. Basu, S., Jacobs, C., and Vanderwende, L. Powergrading: a clustering approach to amplify human effort for short answer grading. *TACL 1* (2013), 391–402.
2. Gaudencio, M., Dantas, A., and Guerrero, D. D. Can computers compare student code solutions as well as teachers? In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, ACM (New York, NY, USA, 2014), 21–26.
3. Huang, J., Piech, C., Nguyen, A., and Guibas, L. J. Syntactic and functional variability of a million code submissions in a machine learning mooc. In *AIED Workshops* (2013).
4. Marton, F., Tsui, A., Chik, P., Ko, P., and Lo, M. *Classroom Discourse and the Space of Learning*. Taylor & Francis, 2013.
5. Nguyen, A., Piech, C., Huang, J., and Guibas, L. J. Codewebs: scalable homework search for massive open online programming courses. In *WWW* (2014), 491–502.