\begin{abstract}

When teaching programming or hardware design, it is pedagogically valuable for students to generate nontrivial examples of functions, circuits, and system designs. Teachers can be overwhelmed by these types of student submissions when running large residential and recently released massive online courses. The underlying distribution of student solutions submitted in response to a particular assignment may be nontrivial, but the newly available volume of student solutions represents a denser sampling of that distribution. Based on these large datasets of students solutions, I am building systems with user interfaces that allow teachers to explore the variety of their students’ correct and incorrect solutions. Forum posts, grading rubrics, and autograders can be based on student solution data, and turn massive engineering and computer science classrooms into useful insight and feedback for teachers. In the development process, I hope to describe essential design principles for such systems.

\end{abstract}

\section{Introduction}

When teaching programming or hardware design, it is pedagogically valuable for students to generate nontrivial examples of functions, circuits, and system designs. However, when running large residential and massive online courses, teachers can be overwhelmed by these types of student submissions. Summarizing, exploring, and assessing these types of solutions to assigned problems, even those which can be run through a battery of test cases, involves unsolved challenges.

This work focuses on engineering course assignments that have a behavioral specification students must meet, and allow for a broad range of internal designs. There may be several distinct, correct solutions, some of which may be unanticipated by teachers, making it harder for them to help students reach their own correct solutions.

The underlying distribution of student solutions to a particular assignment may be nontrivial, but the newly available volume of student solutions represents a denser sampling of the distribution. The increasing scale of the classroom creates a research opportunity. For example, if we attempt to classify solutions with a Support Vector Machine (SVM), the volume of labeled training data (solutions labeled by teachers) is key to its performance. Such classifiers could be incorporated into the user interfaces I am designing for teachers to explore students’ solutions, and could even be trained further based on users’ interactions with the interfaces.

I am guided by the following questions:

\begin{enumerate}

\item How do we help teachers understand the space of programming solutions generated by students? What features of solutions, and user interface designs, are useful for visualizing and clustering alternative solutions?

\item How do we help teachers understand whether the students are learning the right things; improve their teaching materials or respond to common problems; improve a specific assignment or the grading scheme; etc.?

\item As a side-effect of discovering common types of solutions, as well as bugs and misunderstandings, how can peer-to-peer teaching and assistance be enhanced by this knowledge?

\end{enumerate}

{\bf Thesis Statement} A system that empowers teachers to explore the variety of their students’ correct and incorrect solutions will enable data-driven refinements to teaching materials. Forum posts, grading rubrics, and autograders can be based on student solution data, and turn massive engineering and computer science classrooms into useful insight and feedback for teachers.

…..

At the intersection of information visualization and program analysis is an interactive visualization embedded in the MathWork’s Cody\footnote{\url{mathworks.com/matlabcentral/cody}}, an informal learning environment for the Matlab programming language. The Cody programming challenge does not have any teaching staff associated with it but does have the interactive {\em solution map} visualization to help participants discover alternative ways to solve the programming problem, after they submit at least one function that passes all test cases. A solution’s parse tree size is the arbitrary metric by which solutions are ranked, and some participants try to beat their own previous submissions, through their own ingenuity and potentially the mining of alternative code snippets from other solutions revealed in the solution map. The solution map plots each solution as a point against two axes: time of submission on the horizontal axis, and parse tree size on the vertical axis. Despite the simplicity of this metric, solution maps can provide quick and valuable insight when assessing large numbers of solutions~\cite{ICERGlassman}.