Expanding Computer Science Education by Developing Scalable Methods for Automated Assessment

Mehran Sahami
Computer Science Department
Stanford University
sahami@cs.stanford.edu

The Bureau of Labor Statistics projects that the need for computer scientists in the coming decade will far outstrip the supply. And it is often lamented that not enough students are trained in computing skills at all levels of the educational pipeline. While the Computer Research Association has reported up-ticks in the past few years in the number of students pursuing computer science degrees at the undergraduate level, these increases are still not sufficient to meet the growing demand for graduates with computing skills. Moreover, at earlier stages in the educational pipeline (i.e., K-12), the lack of computer science instructors has made is very difficult to broaden participation in computing or make computing education widely available. Indeed, as computing becomes an integral component of other fields (e.g., Computational X, where X is another discipline such as Biology or Economics) the need for students to be trained with some significant computing skills (even if not as majors in computer science) will be further exacerbated.

The need to train more students with computing skills at all levels has led to recent efforts to increase the number of K-12 computer science teachers (most notably, the NSF’s CS10K project) as well as the exploration of ways in which recently popularized Massive Open Online Courses (MOOCs) may be employed to create additional capacity with regarding to teaching computer science. While efforts such as these are to be lauded (and studied to determine the real educational outcomes they produce), a critical element of these efforts is making sure that sufficient capacity is available for assessing the work of students in such courses. Currently, human assessment of student work in computing courses is the gold standard. However, various forces make the scaling of human assessment a challenge in trying to significantly increase educational opportunities in computer science. For example, economic forces often pull educators with computer science skills out of teaching and into much higher paying jobs in industry. Also, newly trained educators in computing need time to develop strong assessment skills and would benefit from tools that help to standardize the assessment process and the resulting outcomes. Thus, an opportunity arises for the development of rich, automated assessment methods to be able to evaluate and provide feedback on students’ work in computing
courses. Such automated assessment is especially critical at the introductory level, as these are the courses likely to have the highest enrollments, reaching students from disciplines beyond computing. Automated assessment is especially critical in MOOCs to allow evaluation of student work to more readily scale with the reach of the course, especially in courses with tens of thousands of students.

Indeed, many forms of automated assessment (many of which are not specific to computing) already exist. However, the challenge in such assessment methods—and the research opportunities afforded as a result—is just how rich the space of student work being assessed can be. Here, we specifically consider student work in introductory computer science courses (primarily programming courses), since such courses seem to be at the crux of expanding the pipeline of students with computing skills. As a rough hierarchical classification, we can consider five levels of “richness” in the space of assignments in computing courses that we can consider for evaluation (note that each level subsumes the levels above it):

1. Short answer and multiple choice questions
2. Programs or modules that produced a desired (fixed) output, usual textual
3. Programs or modules that can be unit tested (i.e., have a specified API)
4. Programs that can be evaluated within a particular constrained framework
5. Unconstrained programs

Currently, automated assessment techniques for the first three levels of this hierarchy are generally well understood. Unfortunately, even reaching the third level of richness places significant constraints on the types of assignments students can be given. For example, interactive, graphical programs (such as many games) do not produce a fixed output and also do not lend themselves to robust assessment via unit testing. Yet, it is precisely these sorts of assignments which are likely to spur interest in computing and create a viral enthusiasm among student who wish to show their work to their friends. Moreover, requiring unit testing frameworks to be used with assignments limits the open-endedness of student work and constrains important design choices that students would normally explore in developing their solutions.

The development of robust, automated assessment methods for computer science assignments provides an opportunity to scale computer science education in both the traditional “live” and newly touted MOOC formats. However, developing methods for rich automated assessment will require research from both the domains of computer science and education. From the standpoint
of computer science, significant technical challenges need to be addressed, a short list of which is provided for illustration:

- Methods for analyzing programs to evaluate their execution under specific conditions. Such methods can include static analysis of programs, including the identification of certain programming patterns that are likely to appear in correct solutions.

- Robust methods for determining “similarity” of program segments so as to assess programs with respect to known solutions or analyze work across students. Such measures of “similarity” between programs can also be harnessed in other contexts, such as plagiarism detection, discovery of “clusters” of related solutions, and measures of diversity of a solution space to better understand the multitude of ways that students may approach a given problem.

- Machine learning algorithms for the discovery of patterns in program development. Such algorithms can give us a better understanding not only of how students make progress in generating a solution, but also help discover patterns in student work reflecting common misconceptions about programming. The identification of such patterns, in turn, helps to inform pedagogy that can ameliorate such misconceptions. In this way, a direct link between automated assessment and insights into improving classroom instruction can be forged.

With respect to education, research in automated assessment has the potential to lead to additional benefits, such as:

- Developing a greater awareness of how various types of assignments in introductory programming courses help to develop students’ understanding of particular topics in those courses,

- Insight into how different levels of flexibility in the space of assignments given can be balanced against the ease and accuracy of the automatic assessment of those assignments,

- Determining the extent to which automated assessment methods can also be used to help calibrate and train new teachers in their own assessment of student work.

Research in automated assessment of student work in introductory programming classes provides a significant opportunity to scale the availability of educational computing resources. Moreover, such work helps build a strong link between the technical and educational aspects of work in computer science education. Domain specialists in computing will be needed to devise better algorithms for program analysis and data mining, while educators with expertise in computing will help determine how to best deploy these technologies to have a meaningful impact in
expanding access to computer science education. Ultimately, the research question to be addressed here is whether it is possible to build rich automated assessment methods for student work in introductory computer science classes that can truly provide students (and instructors) with meaningful feedback about student work, while also providing instructors sufficient flexibility in the design of assignments so that they capture the imagination of the students in their courses and showcase the power of computing.

Expanding the pipeline of students in computer science by truly scaling computer science education presents a formidable challenge. However, we have the opportunity to explore the development of technologies that may significantly accelerate our ability to provide computer science education at scale. Research into the development of rich, automated assessment methods for introductory computing courses would be a significant step in that direction. The research challenges of building workable systems for assessment will be great, but the potential pay-off in terms of making rich educational opportunities in computer science available at scale will be far greater.