Teaching Statement

My excitement for teaching comes from my research goal of developing new technologies for improving student learning. When I teach, I try to be rigorous but at the same time I try to make the material fun and interesting. I like to use real-world examples to motivate the material and real-world data sets for making homework assignments more interesting. I strive hard to teach the basic principles using a lot of simple examples until students are comfortable with the main concepts, and then I build on top of them to explain more complex ideas. My teaching philosophy is highly influenced from my teaching experience as a teaching assistant and from student and faculty interactions during my research studies. Being a part of the Indian national junior bridge team, I have also had a chance to teach bridge at the competitive level to many people and I have found similar teaching techniques to work well.

My Teaching Experience

I was a teaching assistant (TA) for the Introduction to Algorithms course at MIT (6.006) in Spring 2009, and the head TA for the same course in Fall 2009. In the end-of-course evaluations, I received ratings of 5.9/7 and 6.2/7 (highest amongst the course staff of two professors and five TAs) for the two terms respectively. My responsibilities were to conduct two recitation sections every week consisting of around 20-30 students each and help with the preparation of lecture materials, assignments, and exams. In my recitation sections, I tried to explain the material taught in the lecture in a different way than it was presented in the class and I also tried to cover a bit more advanced material. Students seemed to appreciate an alternate perspective on the lecture material as one of the anonymous comments said “I learned more in his recitation than in lecture”. My approach was to use lots of basic examples to explain the fundamental concepts first and then build on them to explain more complex algorithms and applications. Students seemed to appreciate this as well; from one of the anonymous comments “Covered a lot of illustrative (going from easy to difficult gradually) problems: the best way to help students gain experience in applying algorithms”.

To make recitations interesting, I tried to explain the material using many real-world problems that students could easily relate to. For example, when I used the currency exchange arbitrage problem\(^1\) to teach them how the Bellman Ford algorithm can be used to find negative cycles in a graph, I could clearly see a much higher interest level of students and class participation. I, together with the course staff, also tried to make problem sets more exciting. For example, we gave students rubik cubes for the problem set that asked them to perform breadth-first search to find shortest path from one rubik cube configuration to another, and we used a large data set of real web results for the problem set that asked students to compare rankings of two search engines using the longest common subsequence algorithm.

My Research Originating from Teaching Experience

My passion for improving student learning has also led me to pursue it as one of my main research goals. I am trying to build new intuitive interfaces to help students learn different concepts as well as systems to provide automated feedback on their exercise attempts.

I found a big disconnect in the way I was teaching data structure algorithms in the classroom using high-level box-and-arrows diagrams and the way students were supposed to implement them in a low-level language. I built a system, called Storyboard Programming Tool (SPT) [3], that tries to bridge this gap. SPT lets students describe data structure manipulations using visual input-output diagrams (similar to the same way they are taught in the classrooms) and it automatically synthesizes low-level pointer manipulating code from them. I am collaborating with Andrew Correa (a PhD student at MIT) in pedagogical studies to evaluate how SPT helps students in learning data structure manipulations.

---

\(^1\)Given a set of currency conversion rates, find if there is a way to make infinite profit by buying and selling currencies.
As a teaching assistant, I used to spend many hours every week to go through the homework code submissions to provide students feedback and partial credit for their attempts. I built a system called Autograder [1] that automatically generates such feedback for student assignments describing exactly what is wrong with their solution and how to correct it. I have been working with the TAs of the Introduction to Programming course to integrate the system in the grading workflow. I have also discussed this system with many faculty who are teaching this course at MIT and edX, and they are excited to explore ways to combine it with the teacher’s knowledge for improving student learning. I am currently designing an A/B test to deploy on the edX platform to evaluate the pedagogical value of Autograder’s feedback. I am also collaborating with Elena Glassman (a PhD student at MIT) to build a similar system for the Introduction to Digital Circuits course.

Advising Undergraduate Researchers

I strongly believe that undergraduate education should include some research component to broaden student’s horizons and to help them know current state-of-the-art techniques that they can apply in their careers. I was fortunate enough to work with 5 undergraduate students through MIT’s Undergraduate Research Opportunities Program (UROP). They worked on research projects such as extending Autograder to support strings and classes, using Storyboard Programming to synthesize heap and graph algorithms, and performing Hindley-Milner style type inference for python programs. Our work with one of the UROP on doing modular synthesis in Sketch using function models was recently published at the VMCAI conference [2]. During the course of advising these students, I learned that they were routinely under heavy course load and didn’t have that much time to perform exploratory research. This made me narrow down the scope of their projects and break it down into very concrete goals. I found this approach worked particularly well for students to make progress on their projects.

Teaching Interests

At the undergraduate level, I am well equipped to teach courses such as Introduction to Programming, Introduction to Algorithms, Programming Languages, Compilers, Logic in Computer Science, and Formal Languages and Automata Theory. I am interested in using tools from my research to help students learn the materials more effectively. For the introductory programming course, I would like to develop a series of assignments where each assignment would involve building some component of a game so that at the end of the course they would have developed a complete game that they can share with their friends and families. I believe this will generate excitement even in non computer science students to take more computing courses. I would also like to create an online programming interface using the Autograder system, which the students can use to practice and learn different programming concepts while getting automated feedback. For the Algorithms course, I would like to use SPT to enable students to primarily focus on the basic concepts of data structures without having to worry about the intricate low-level programming details.

At the graduate level, I will be excited to teach courses such as Program Analysis Techniques and Advances in Program Synthesis. In the program analysis course, I would like to teach the concepts of specification logics, model checking, abstract interpretation, testing, and static and dynamic analysis. I believe these areas have matured a lot in the past few decades and are starting to prove useful in many research areas other than formal methods and software engineering. In the program synthesis course, I would like to teach students both the classical approaches of program synthesis as well as recent breakthroughs and new applications. This will enable students to know about the current state-of-the-art and perform cutting-edge research on automated program synthesis.
References

