Teaching Statement

Kevin Matulef

December 1, 2013

Teaching has been an indispensable part of my entire career, from my early days as an undergraduate, to my current ones as a postdoc. Being able to share my passion for computer science, and ignite that same passion and curiosity in others, has consistently served as a source of personal motivation. Indeed, one of my primary reasons for seeking an academic position is for the opportunity to teach and learn from students.

Over the course of my career, I have taught at four different universities, located on three different continents. Each of these institutions had a very different atmosphere, and I have learned different lessons from teaching at each. Below, I summarize some of my experiences, and discuss how they have shaped my teaching philosophy.

Teaching Philosophy

The primary goal that I strive for is to teach in way that ensures students are *engaged and actively contributing to their own education and that of their peers.* Learning is not just about memorizing facts, theorems, and proofs, but about internalizing the ideas, figuring out why they are important and applicable, and communicating them to others. As any teacher will attest, it is often this last part- communicating ideas to others- that gives one a full understanding of the material. Thus, I believe a good teacher not only communicates the material to the students, but *leaves them with the desire and ability to teach it themselves.* Of course, there is no single "magic bullet" for engaging students in this way. Instead, I believe it calls for a variety of approaches, both traditional and non-traditional:

- **Give students the opportunity to be teachers.**

  I learned this lesson early, when I was just an undergraduate at Brown University. Brown is unique among the institutions I have been a part of, in that it employs vast numbers of undergraduate students as TAs. In my sophomore and junior years, I served as a TA (and later the "head TA") for a year-long introductory computer science course. Even though I was only an undergraduate, at Brown I had much more responsibility than a typical TA would at other universities. I got to help shape the course by coming up with new assignments, leading a weekly "lab" session where students completed short programming exercises, and even giving some guest lectures. As Head TA, I additionally managed a staff of 6-9 others.

  Brown's undergraduate TA program clearly demonstrated to me how much students can learn when given the opportunity to be teachers themselves. For me personally, the program increased my enthusiasm and mastery of the material, and as a relatively shy undergraduate it gave me a reason to interact with professors who later served as mentors. The TA program had an even larger affect on the community as a whole, since it helped to create an atmosphere where students constantly asked each other questions and discussed the material with each

other. I personally know many other Brown alumni who have gone on to successful careers and directly credit the TA program with developing their skills.

As a professor, I plan to give my own eager students the same opportunities to help teach. I don't expect this will reduce my workload, since supervising TAs requires a lot of attention, and the main work of planning the course content and lectures is still no small task. But I expect the students will gain a lot from the experience.

Ideally, I hope to foster a large TA program, such as the one at Brown, but I realize that at some universities this may be impractical. In this case, it is still possible to help students develop the same sort of skills, by giving them assignments and administering exams in a way that forces them to *think* like teachers.

- **Give exams and assignments that encourage students to communicate ideas.**

  A second teaching lesson I've learned is that it is possible to encourage students to think like teachers even in an exam setting. In Denmark, where I co-taught (with Joshua Brody) a graduate-level course on Property Testing, nearly every university course has an oral final exam instead of a written one. The exams are short, around 15-20 minutes. Students are told that they will be asked to present one topic from a predefined list (a 12-week course might have, say, 12 topics), chosen at random when they arrive for the exam. During their presentation, the examiners can ask probing questions.

  Although I was initially skeptical of the oral exam format, after administering such an exam myself, I became convinced of its merit. During the exam for my own course, I was surprised at how well the Danish students performed; one student gave a beautiful explanation of a Property Testing algorithm that I believe surpassed my own. Based on my experience, I believe the exam format effectively incentivizes students to hone their communication skills and think critically about the material.

  At Tsinghua University, I taught a full undergraduate course on Theory of Computation, and one of my regrets is that I did not administer oral exams or give any assignments that encouraged students to practice communicating the material. Although I received good teaching reviews, I ran the course in a traditional way, and assigned traditional homework problems. The students were highly talented, but I believe their communication skills were lacking, largely due to lack of practice (the Chinese educational system typically emphasizes written assignments and exams). Administering oral exams, or at least asking students to prepare short presentations for the class, might have helped remedy this.

- **Prepare lectures with lucid examples and provocative questions**.

  Traditional lectures still serve an important role in university education. They keep students on a schedule, provide them with opportunities to interact and ask questions in person, and of course they serve as a "first pass" over the material. While I am intrigued by the rise of Massively Open Online Courses (MOOCs), I do not think they will replace traditional lectures anytime soon; instead, I think they will co-exist and complement them. So, while it may be obvious, it is still worth pointing out that good teachers cannot neglect lecture preparation.

  Giving a good lecture involves so many different elements- knowing your audience, having good blackboard technique, showmanship, etc.- that it is impossible to articulate them all here. But I will point out one element that I always keep in mind: the value of asking provocative questions. A good lecture tells a story, and I like to interrupt that story intermittently

to have the students predict where it will go. This can keep their attention by employing an element of surprise (e.g. "min-cut is polynomial time, so max-cut obviously is too, right? Hmm, not so fast..."). Asking provocative questions, while simultaneously providing lucid and clear examples, can be an effective way of keeping students engaged, and making them think about how *they* would present the material.

I have already acquired much experience as a lecturer, having taught an undergraduate class at Tsinghua, co-taught graduate-level classes at Tsinghua and Aarhus, and TA'd two courses with recitation sections at MIT. By preparing my lectures carefully, and punctuating my lectures with provocative questions, I've consistently received high teaching reviews. At MIT, for instance, the students rated me 6.1/7 and 6.4/7, and MIT's "Underground Guide" summarized the student evaluations as follows: *"[TA was] very helpful and understood the material well. He was organized and able to explain even difficult concepts,"* and *"TA gave well prepared recitations. He was clear and patient."*

Future Plans

Although I have performed well as a teacher so far, I believe I still have much to learn. As a full-time faculty member, one of the things I look forward to is the opportunity to experiment with different teaching techniques, and to develop new courses myself.

My background in mathematics qualifies me to teach a large number of courses in theoretical computer science, such as *discrete mathematics, probability, algorithms, data structures, computational complexity, and cryptography.* For any of these subjects, I would be comfortable teaching either an introductory level course, or a more advanced course for graduate students or upper-level undergraduates. I already have some ideas for my own versions of some of these courses. For instance, rather than teaching an "Advanced Complexity" class, I'd like to teach a class that covers much of the same material, but from the perspective of the techniques and false proofs that we know *don't* work to resolve our field's biggest open questions. Such a class could be called "Great Failures of Theoretical Computer Science," but of course it wouldn't just be about failure; it would really be an entertaining way to introduce some advanced techniques in complexity theory, and explore their power as well as their subtle limitations.

I would also enjoy teaching courses that are not precisely in my area of expertise, but are closely related, such as *databases* or *machine learning.* I spent a year working for MongoDB Inc., and that experience gave me a sense of the real-world database problems people encounter in industry. I think there is value in teaching such a course with some added theoretical components (for instance, covering some distributed algorithms and I/O-efficient algorithms). Preparing a course outside of my area would obviously involve a lot of work, but I would enjoy the challenge. I would also enjoy co-teaching such a course with another professor whose area of expertise complements my own.

Finally, I would enjoy teaching an introductory computer science course, aimed at majors and non-majors alike, teaching the basics of programming alongside the basics of algorithmic theory. Such a course holds a special place in my heart. Before my freshman year of college, I had never taken a computer science class before, and indeed, I had no intention of majoring in it. But I still remember the excitement I felt when I took the intro course taught by Phil Klein and Leslie Kaebling. It was there that I learned computer science was not just about twiddling bits, but about exploring what is and is not computable in the world, and about utilizing the tool of computation to solve problems affecting virtually every aspect of humanity. I can think of few things more exciting than imparting that same knowledge onto future generations of students.