Teaching Statement

Benjamin Berg

Teaching Experience

Over the last ten years, in addition to studying computer science, I have been teaching and working to improve educational experiences in computer science.

Wide array of teaching experience in CS Theory and Systems. My earliest teaching experience was as an undergrad teaching assistant at Duke University, where I was a teaching assistant for three different theory and systems classes over four semesters. Then, before entering graduate school, I spent a year at the MIT Media Lab in the Lifelong Kindergarten group, which focuses on developing better computer science learning experiences for students of all ages. The ethos of this research group, which has heavily influenced my teaching philosophy, is that a wider range of students can be reached through the design of *creative learning experiences* which stress learning through engaging with concepts via personally meaningful projects. As a graduate student, I served as a teaching assistant for the performance modeling of computer systems course and the computer architecture course at Carnegie Mellon University (CMU). I received an average overall rating of 4.81 out of 5 on my teaching reviews from students. Reviews noted my "...ability to hear your question, instantly understand the exact source of your misunderstanding, and to ask exactly the right question to help you understand the material better," and mentioned that I "...was always willing to walk through even the most basic steps of solving a problem and was very approachable during office hours."

My responsibilities across these various teaching roles have varied widely, giving me extensive experience in the design and implementation of successful computer science courses. I have taught lab sections, held office hours, performed code reviews, written exams, facilitated workshops, and led lectures. I have therefore developed a teaching approach that is informed by my first hand experience with the successes and failures of different teaching strategies across multiple subjects, experience levels and age groups. In pursuing a faculty position, I am excited to use these lessons learned and to continue to hone my abilities as an instructor.

Mentoring and advising experience in CS Theory and Systems. In addition to teaching courses, I am passionate about mentoring students and advising student research. I have been very lucky during my time at CMU to advise several talented undergraduate and masters students. I advised Corwin De Boor, an undergraduate, while he worked on a theoretical scheduling problem, resulting in the presentation of his research at the CMU Undergraduate Research Symposium. I also helped advise a masters student, Justin Wang, who completed a research project on caching that resulted in a poster presentation at SIGMETRICS in 2018. I am also currently working with a student, Deepayan Patra, to design an improved scheduler for parallelizable database queries. Deepayan started working on this project as an undergraduate, and he will be returning to CMU to continue this work as a masters student this year. Finally, I am currently advising six students at Columbia University who are taking a course on Distributed Storage Systems. These students are completing semester-long research projects using the CacheLib caching engine that I worked on with Facebook. The students are using CacheLib to evaluate state-of-the-art cache workload generation methods, and the hope is that these projects will serve as preliminary work for a paper on workload generation.

Teaching Methodology

In my experience, many college-level classes are either too easy or too hard for the majority of students in the class. In some cases, courses are targeted towards students with less background or interest in the course material. In these cases, students are not pushed to think deeply about the core concepts of the course. In other cases, the material is targeted towards students who specialize in the course topics and enter the classroom with the most background knowledge. Outside of a limited number of highly-specialized courses, this option can be even worse than the first. While specialists can excel in these courses, nonspecialists see that the material is not targeted to them and reciprocate by not engaging with the material. Even more insidiously, *these courses push nonspecialists to avoid certain topics or entire fields of study altogether.*

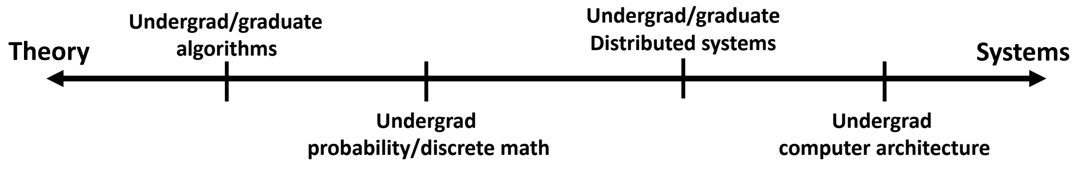
Learn by teaching. The problem is that nonspecialists need additional opportunities to engage with material while specialists must be challenged to understand the material at a deeper level than what might be practical for a class lecture. My solution is to place students in mixed-ability groups where they will discuss and collaborate on coursework. This group work provides additional opportunities for nonspecialists to ask questions and clarify their understanding of concepts they have not grasped during lecture or independent review. Specialists are simultaneously challenged to organize their understanding of the material well enough to explain it to their peers. As many university faculty members will readily admit, thorough understanding of complex concepts often comes from teaching these concepts to others. Ironically, this sort of teaching experience is often scarce for students, particularly at the undergraduate level. Hence, in addition to deepening their understanding of the course material, specialists will develop teaching skills that will be essential as they head into industry jobs or graduate programs in computer science.

Make everyone a specialist. The crucial second step of my methodology is to have students apply the course material to the subject areas which excite them the most. The key observation here is that the specialist/nonspecialist designation is often superficial, and can be uncorrelated with a student's long-run aptitude in a subject area. Encouraging students to work on personally interesting application areas quickly breaks down this designation. Students who considered themselves to be nonspecialists can find themselves explaining the intricacies of a particular application to their peers who had previously been leading group discussions. For example, when I taught CMU's graduate performance modeling course, students came from computer science theory, systems, computer engineering, and operations research (OR) backgrounds. The beginning of the course requires students to come up to speed on applied probability and stochastic processes, material that tends to be familiar to theory and OR students. During these first weeks, I used recitation to have these students help their peers review the background material. Then, the course turns its attention to modeling and analyzing real-world systems. Suddenly, the systems and engineering students became experts who could explain their intuition from years of first-hand experience. Introducing various application areas ensures that every student has an opportunity to act as both a specialist and nonspecialist.

Learn via debate. My methodology works best when students can disagree on an answer, debate the validity of two arguments, and collaboratively reach an answer. This debate is crucial to my methodology because *I feel that mastery of a subject comes from the ability to evaluate competing claims in the area.* The discussion and debate process provides students much needed practice in this essential skill. When teaching performance modeling, for example, such debates became a common occurrence in office hours and recitations. Small groups of students would work together on handouts or problem sets. When there was disagreement between group members, I would encourage students to present their answers to each other on the board and examine the places their solutions differed. Regardless of who was ultimately correct, the insight gained from these debates was far greater than if students had simply turned in a correct or incorrect solution to the problem. Providing this deeper conceptual understanding of the material is the ultimate payoff of my teaching methodology.

Courses I Will Teach

Given that my research lies at the intersection of theory and systems and that I have deep experience in both areas, I feel completely comfortable teaching undergraduate and graduate classes in either area. Some examples of courses I could teach are shown in the figure below.


Theory
Undergrad/graduate algorithms
Undergrad probability/discrete math
Undergrad/graduate Distributed systems
Undergrad computer architecture
Systems

New Courses. In addition to the above courses, I would look to create three new courses. First, I would love to teach an undergraduate or graduate class in parallel computing and architecture. This would be a systems class focused on building and executing fast parallel code on modern hardware, and would include both a survey of parallel computer architecture and a quantitative modeling component. Second, I would be eager to teach a performance modeling class geared towards both systems and theory students. This course would teach the foundations of stochastic performance modeling and use these tools to analyze a variety of real-world systems. Finally, I would like to develop a graduate course in parallel scheduling. This would be a theory course for graduate students who have a background in scheduling and queueing. The course would cover new analytical techniques and open research problems from the performance modeling community.